```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

void TableWrite(unsigned char *dest, unsigned char *source, unsigned short Count);
void TableRead(unsigned char *dest, unsigned char *source, unsigned  short Count);
```

```
/*\
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"

#ifdef IR_RULES

#ifdef PIC
#include "delay.h"
#include "i2c_ccs.h"
#include "tablereadwrite.h"
#else
#include <io.h>
#include <fcntl.h>
#endif

#include "sendircommon.h"
#include  "sendirrules.h"
#include "beep.h"
#ifdef DEBUG
#include <stdio.h>
#endif
#include <string.h>
#include "fsdtablelarge.h"
#include <ctype.h>


short                       devTicks;
extern short      irScriptBuffer;



void ir_initDevice(void)
{
         NodeId nodeDevice;
         char buffer[4];

         fsd_switchRomBuffer(irScriptBuffer);
         nodeDevice = fsd_getRootNode();
         if (nodeDevice != NODE_ERROR) {

                  fsd_getAttribute(nodeDevice, "ticks", buffer, 4);
                  devTicks = (short)atoi(buffer);
         }
         else {
                  devTicks = -1;
         }
         debugHi(("devTicks %d node %d", devTicks, nodeDevice));
         ir_rulesInit();

         fsd_unswitchRomBuffer();
         return;
}



void ir_LedOn(const unsigned short T)
{
```

```
#ifdef PIC
        IR_LED_ON;
        DelayBigUs(T);
#ifndef IR_RULES
        IR_LED_OFF;
#endif
#endif
}


void ir_LedOff(const unsigned short T)
{

#ifdef PIC
#ifdef IR_RULES
        IR_LED_OFF;
#endif

        DelayBigUs(T);
#endif

}



void ir_Initialize(void)
{
        struct eprom_script_def script;
        short scriptType, scriptId;

        devTicks = -1;
        scriptType = IRSCRIPT;
        if (epromValid() ) {
                scriptId = epromReadWord(EPROM_IR_SCRIPTID);
        }
        else {
                scriptId = -1;
        }

        if (scriptId != -1) {
                if (epromGetScript(scriptType, scriptId, &script) == -1) {
                        fsd_setScriptBuffer(scriptType, scriptId);
                } else {
                        fsd_setScriptBufferNoLoad(&script);
                }


                ir_initDevice();
                if (devTicks == 0) devTicks = -1;
#ifdef IR_RULES
                if (devTicks != -1) {

                        fsd_switchRomBuffer(irScriptBuffer);

                        if (epromGetScript(IRDATA, -1, &script) == -1) {
                                ir_configIrCodes();

                        }
                        else {
                                ir_configIrCodesRom();
                        }
                        fsd_setMainScriptBuffer();
                }
#endif
        }
```

```
        if (devTicks == -1) {
                errorBeep();
                debugPutstrHi("No ir device");
        }

}



long ir_CalcFrequency(const short N)
{
        long ret;

        ret = (long)(PRONTOFREQUENCY/(103 * (float).241246));
        return ret;
}



short ir_CalcOneCycle(const long frequency)
{
        float x;
        short ret;

        x = ((float)1 / frequency);
        x += (float).0000005;
        ret = (short)(x * 1000000L);
        return ret;
}

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include <pic18.h>
#include "delay.h"
#ifdef DEBUG
#include "serial.h"
#include <stdio.h>
#endif
#include <string.h>
#include "i2c_ccs.h"



short ROM_ReadWord(int address)
{
        short ret;
        random_readM(0x00, address, (char *)&ret, 2);
        return ret;
}







void ROM_Send(int Address, char *Data, char Num)
{
         while(Num--)
         {
              random_write(0x00, Address, *Data);
              Data++;
              Address++;
         }
}



void ROM_Read(int Address, void *Data, char Num)
{
        random_readM(0x00, Address, Data, Num);
}
```

```
void random_write(char dev_adr, int mem_adr, char dat)
{
   i2c_start();
   i2c_out_byte(0xa0 | (dev_adr << 1));
   i2c_nack();
   i2c_out_byte((mem_adr >> 8) & 0xff);
   i2c_nack();
   i2c_out_byte(mem_adr & 0xff);
   i2c_nack();
   i2c_out_byte(dat);
   i2c_nack();
   i2c_stop();
   DelayMs(25);
}

void random_readM(char dev_adr, int mem_adr, void *Data, char Num)
{
   char i;
   char *p=Data;

   i2c_start();
   i2c_out_byte(0xa0 | (dev_adr << 1));
   i2c_nack();
   i2c_out_byte((mem_adr >> 8) & 0xff);
   i2c_nack();
   i2c_out_byte(mem_adr & 0xff);
   i2c_nack();
   i2c_start();
   i2c_out_byte(0xa1 | (dev_adr << 1));
   i2c_nack();
   for (i=0; i < Num; i++) {
            *p++=i2c_in_byte();
            if (i != Num - 1) {
                  i2c_ack();
            }
   }
   i2c_stop();
}

char random_read(char dev_adr, int mem_adr)
{
   char y;
   i2c_start();
   i2c_out_byte(0xa0 | (dev_adr << 1));
   i2c_nack();
   i2c_out_byte((mem_adr >> 8) & 0xff);
   i2c_nack();
   i2c_out_byte(mem_adr & 0xff);
   i2c_nack();
   i2c_start();
   i2c_out_byte(0xa1 | (dev_adr << 1));
   i2c_nack();
   y=i2c_in_byte();
   i2c_stop();
   return(y);
}



char i2c_in_byte(void)
{
   char i_byte, n;
   i2c_high_sda();
   for (n=0; n<8; n++)
   {
      i2c_high_scl();
```

```
        if (SDA_PIN)
        {
           i_byte = (i_byte << 1) | 0x01;
        }
        else
        {
           i_byte = i_byte << 1;
        }
        i2c_low_scl();
    }
    return(i_byte);
}

void i2c_out_byte(char o_byte)
{
    char n;
    for(n=0; n<8; n++)
    {
        if(o_byte&0x80)
        {
           i2c_high_sda();
        }
        else
        {
           i2c_low_sda();
        }
        i2c_high_scl();
        i2c_low_scl();
        o_byte = o_byte << 1;
    }
    i2c_high_sda();
}

void i2c_nack(void)
{
    i2c_high_sda();
    i2c_high_scl();
    i2c_low_scl();
}

void i2c_ack(void)
{
    i2c_low_sda();
    i2c_high_scl();
    i2c_low_scl();
    i2c_high_sda();
}


void i2c_start(void)
{
    i2c_low_scl();
    i2c_high_sda();
    i2c_high_scl();
    i2c_low_sda();
    i2c_low_scl();
}

void i2c_stop(void)
{
    i2c_low_scl();
    i2c_low_sda();
    i2c_high_scl();
    i2c_high_sda();

}
```

```
void i2c_high_sda(void)
{

   SDA_DIR = 1;

}

void i2c_low_sda(void)
{
   SDA_PIN = 0;
   SDA_DIR = 0;

}

void i2c_high_scl(void)
{
   SCL_DIR = 1;

}

void i2c_low_scl(void)
{
   SCL_PIN = 0;
   SCL_DIR = 0;

}
```

```
'
 ' PushPlay -- An Xml Document emulator\interpreter for microprocessors
 ' Copyright (C) 2002, Arthur Gravina. Confidential.
 ' Arthur Gravina <art@agravina.com>
 '
 '


VERSION 5.00
Begin VB.Form Form1
   Caption         =   "Compile Ir Codes"
   ClientHeight    =   7095
   ClientLeft      =   60
   ClientTop       =   450
   ClientWidth     =   10185
   LinkTopic       =   "Form1"
   ScaleHeight     =   7095
   ScaleWidth      =   10185
   StartUpPosition =   3
   Begin VB.CommandButton cmdCompile
      Caption         =   "Compile"
      Height          =   495
      Left            =   7920
      TabIndex        =   4
      Top             =   720
      Width           =   2055
   End
   Begin VB.DriveListBox drvList
      Height          =   315
      Left            =   120
      TabIndex        =   3
      Top             =   360
      Width           =   3495
   End
   Begin VB.DirListBox dirList
      Height          =   2790
      Left            =   120
      TabIndex        =   2
      Top             =   1080
      Width           =   3495
   End
   Begin VB.FileListBox filList
      Height          =   3405
      Left            =   4080
      MultiSelect     =   1
      TabIndex        =   1
      Top             =   360
      Width           =   3255
   End
   Begin VB.ListBox List1
      Height          =   1815
      Left            =   360
      TabIndex        =   0
      Top             =   4800
      Width           =   7695
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit
Private WithEvents oCompiler As FSDCompileScript
Attribute oCompiler.VB_VarHelpID = -1

Private Sub cmdCompile_Click()
```

```
    compileFiles
End Sub

Private Sub dirList_Change()

    filList.Path = dirList.Path
End Sub

Private Sub Form_Load()
   Form1.Show
    filList.Pattern = "*.xml"

End Sub

Sub compileFiles()
    Dim inFilename As String, mypath As String, outFilename As String
    Dim ret As Integer
    Dim errors As String
    Dim ind As Integer

    On Error GoTo errrtn
    Set oCompiler = New FSDCompileScript


    If filList.ListCount Then
        mypath = dirList.Path + "\"
        For ind = 0 To Form1!filList.ListCount - 1
            inFilename = Form1!filList.List(ind)
            outFilename = Left(inFilename, Len(inFilename) - 4)
            outFilename = outFilename & ".fsd"
            List1.AddItem "compiling.. " & inFilename
            ret = oCompiler.fsd_loadScript(mypath & inFilename, errors)
            If ret = False Then
                MsgBox errors, vbCritical, "compileCodes WARNING!"
                GoTo errrtn
            End If
            oCompiler.fsd_Compile mypath & outFilename
        Next ind
    End If

    Exit Sub
errrtn:
    MsgBox "compileFiles Error: " & Error

End Sub

Sub compileIrFiles()
    Dim inFilename As String, mypath As String, outFilename As String
    Dim ret As Integer
    Dim errors As String

    On Error GoTo errrtn
    Set oCompiler = New FSDCompileScript

    mypath = "c:\smarttoy\compile ir codes\"
    inFilename = Dir(mypath & "*.xml")
    Do While inFilename <> ""
        List1.AddItem "loadscript.. " & inFilename
        outFilename = Left(inFilename, Len(inFilename) - 4)
        outFilename = outFilename & ".fsd"
        ret = oCompiler.fsd_loadScript(mypath & inFilename, errors)
        If ret = False Then
            MsgBox errors, vbCritical, "compileIrCodes WARNING!"
            GoTo errrtn
        End If
        oCompiler.fsd_Compile mypath & outFilename
        inFilename = Dir
```

```
    Loop
    Exit Sub
errrtn:
    MsgBox "compileIrFiles Error: " & Error
End Sub
Sub compileIrFilesOld()
    Dim fileName As String, mypath As String
    Dim ret As Integer
    Dim errors As String


    On Error GoTo errrtn
    Set oCompiler = New FSDCompileScript
    mypath = "c:\smarttoy\compile ir codes\"
    fileName = mypath & "irCodes.xml"
    List1.AddItem "loadscript.. " & fileName
    ret = oCompiler.fsd_loadScript(fileName, errors)
    oCompiler.fsd_Compile mypath & "irCodes.fsd"
    If ret = False Then
        MsgBox errors, vbCritical, "compileIrCodes WARNING!"
        GoTo errrtn
    End If
errrtn:
End Sub
Private Sub oCompiler_info(sMsg As String)

End Sub

Private Sub info(msg As String)
    Dim obuf As String

    obuf = msg & " " & Now
    List1.AddItem obuf
    List1.ListIndex = List1.ListCount - 1


End Sub
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef __istack_h_
#define __istack_h_

#define MAXDIM 20
#define ISTKERROR -3333
typedef short ElementType;


void IPush(const ElementType f);

ElementType IPop(void);

ElementType IPeek(const ElementType Item);

short ICount();

void EmptyIStack(void);

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef __fsdtablelarge_h_
#define __fsdtablelarge_h_

#include "support.h"
#include  <stddef.h>
#include "eprom.h"

#define NUMSCRIPTS 3
#define NUMDYNAMICNODES 5
#define NUMDYNAMICATTRIBUTES 10
#define NUMDYNAMICTEXTCHUNKS 20

#define TEXT_CHUNK CHAR_BUFFERSIZE
#define SIZETEXTBUFFER TEXT_CHUNK * NUMDYNAMICTEXTCHUNKS

#define NODE_AVAILABLE (WORD) 0
#define NODE_ALLOCATED (WORD) 0x7000


#define NODE_ROOT   -1
#define NODE_ELEMENT 1
#define NODE_ATTRIBUTE 2
#define NODE_TEXT 3
#define NODE_COMMENT 8
#define NODE_EMPTY -1
#define TEXTLOC_EMPTY -1

#define CHAR_BUFFERSIZE 24

typedef struct node_def                    Node;
typedef struct attribute_def        Attribute;
typedef struct node_def                   *PtrNode;
typedef struct attribute_def       *PtrAttribute;
typedef WORD                                NodeId;
typedef WORD                                TextLoc;
typedef char                               *PtrTextLoc;



struct control_def {
        WORD nextLocation;
        WORD numberScripts;
};

#define NEXTLOCATION                offsetof(struct control_def, nextLocation)
#define NUMBERSCRIPTS               offsetof(struct control_def, numberScripts)




struct script_def {
        WORD type;
        WORD id;
        WORD location;
```

```
};

#define NOSCRIPT 0
#define MAINSCRIPT 1
#define IRSCRIPT 2
#define IRDATA 3

#define IRGETSCRIPTID 28001



struct header_def {

        WORD nodeOffset;
    WORD numNodes;
    WORD attributeOffset;
    WORD numAttributes;
    WORD textAreaOffset;
    WORD lenTextArea;
        WORD scriptType;
        WORD scriptId;
};

#define NODEPARENT                  offsetof(struct node_def, parentnode)
#define TYPENODE            offsetof(struct node_def, typenode)
#define NEXTNODE            offsetof(struct node_def, nextnode)
#define FIRSTCHILD                  offsetof(struct node_def, firstchild)
#define FIRSTATTRIBUTE      offsetof(struct node_def, firstattribute)
#define NODENAME            offsetof(struct node_def, locname)
#define NODENAMELEN                 offsetof(struct node_def, lenname)



struct node_def {
    WORD parentnode;
    WORD typenode;
    WORD nextnode;
    WORD firstchild;
    WORD firstattribute;
    WORD locname;
        unsigned char lenname;
        unsigned char filler;
};

#define ATTRIBUTEPARENT             offsetof(struct attribute_def, parentnode)
#define NEXTATTRIBUTE               offsetof(struct attribute_def, nextattribute)
#define ATTRIBUTENAME               offsetof(struct attribute_def, locname)
#define ATTRIBUTEVALUE              offsetof(struct attribute_def, locvalue)
#define ATTRIBUTENAMELEN    offsetof(struct attribute_def, lenname)
#define ATTRIBUTEVALUELEN   offsetof(struct attribute_def, lenvalue)


struct  attribute_def {
    WORD parentnode;
    WORD nextattribute;
    WORD locname;
    WORD locvalue;
        unsigned char lenname;
        unsigned char lenvalue;
};



void fsd_Initialize(void);
```

```
void *fsd_fetchTextLocPtr(const TextLoc locText);
NodeId fsd_fetchNode(PtrNode pNode, NodeId node);
NodeId fsd_fetchNodeId(const NodeId node, const short offset);
TextLoc fsd_fetchNodeTextLoc(const NodeId node, const short offset);
NodeId fsd_fetchAttribute(PtrAttribute pAttribute, NodeId attribute) ;
NodeId fsd_fetchAttributeId(const NodeId attribute, const short offset);
TextLoc fsd_fetchAttributeTextLoc(const NodeId attribute, const short offset);
void fsd_fetchText(TextLoc textLoc, short textLen, char *buffer, const short len);
NodeId fsd_slotNode(void);
void fsd_scratchNode(const NodeId nodeId);
NodeId fsd_slotAttribute(void);
void fsd_scratchAttribute(const NodeId nodeId);
TextLoc fsd_slotTextBlock(void);
void fsd_scratchTextBlock(const TextLoc loc);
TextLoc fsd_addText(const char *sText);

void fsd_getText(const TextLoc locText, char *buffer, const short len);
void fsd_setNodeName(const NodeId node, const NodeId parent, const char *name);
NodeId fsd_getRootNode(void);
short fsd_getChildCount(const NodeId parentNode);
short fsd_getChildNodes(const NodeId parentNode, NodeId nodesFound[], const short len);
NodeId fsd_getChildByPos(const NodeId parentNode, const short pos);
void fsd_getNodeName(const NodeId nodeId, char *buffer, const short len);
short fsd_getNodesByName(const NodeId parentNode, const char *sName, NodeId nodesFound[], con
short fsd_getAttributes(const NodeId parentNode, NodeId nodesFound[], const short len);
short fsd_getAttributeCount(const NodeId parentNode);
NodeId fsd_getAttributeByName(const NodeId parentNode, const char *sName);
NodeId fsd_getAttributeByPos(const NodeId parentNode, const short pos);
void fsd_getAttributeValue(const NodeId attributeId, char *buffer, const short len);
void fsd_getAttribute(const NodeId parentNode, const char *attribName, char *buffer, const sh
BOOL fsd_hasAttributes(const NodeId nodeId);
BOOL fsd_hasChildNodes(const NodeId nodeId);
NodeId fsd_setAttribute(const NodeId parentNode, const char *name, const char *value) ;
short fsd_getInteger(const char *value);
```

```
void fsd_switchRomBuffer(short newRomBuffer);

void fsd_unswitchRomBuffer();

void fsd_setMainScriptBuffer(void);


void fsd_setScriptBuffer(short scriptType, short scriptId);
void fsd_setScriptBufferNoLoad(struct eprom_script_def *script);
#ifndef PIC

void fsd_readRom(short offset, short numBytes);
void fsd_writeRom(short offset, short numBytes);

int fsd_readfile(short scriptType, short scriptId);
#endif

void fsd_LoadMainScript(void);

void fsdint_GetIrScript(void);

void fsdint_SetIrScript(short scriptId);

void fsd_clearEpromScript(short scriptType, short scriptId);
#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#ifdef IR_RULES
#include "fsdtablelarge.h"
#include "sendircommon.h"
#include "sendirrules.h"
#ifndef PIC
#include "pcromchip.h"
#endif

static void defineButton(struct ir_remote*remote, struct ir_ncode *codes, const char *commandI

extern short      offsetFlashMemory;
extern short      numScriptBuffers;
extern unsigned short      scriptBuffer[];
extern short      offsetFlashMemory;

static struct ir_remote remote;

const struct flaglist all_flags[] = {
        {"RC5",                    RC5_CODE},
        {"RC6",              RC6_CODE},
        {"RCMM",             RCMM},
        {"SHIFT_ENC",    SHIFT_ENC},
        {"SPACE_ENC",    SPACE_ENC},
        {"REVERSE",      REVERSE},
        {"NO_HEAD_REP", NO_HEAD_REP},
        {"NO_FOOT_REP", NO_FOOT_REP},
        {"CONST_LENGTH",CONST_LENGTH},
        {"RAW_CODES",    RAW_CODES},
        {"REPEAT_HEADER",    REPEAT_HEADER},
        {"SPECIAL_TRANSMITTER",    SPECIAL_TRANSMITTER},
        {NULL,0},
};

const struct flaglist allCommands[] = {
        {"TITLE",                          TITLE},
        {"MENU",                   MENU},
        {"PLAY",                   PLAY},
        {"STOP",                   STOPDVD},
        {"PAUSE",                          PAUSE},
        {"STEP",                   STEP},
        {"PREVCHAPTER",                    PREVCHAPTER},
        {"NEXTCHAPTER",                    NEXTCHAPTER},
        {"SEARCH",                         SEARCH},
        {"NAV_UP",                         NAV_UP},
        {"NAV_DOWN",                       NAV_DOWN},
        {"NAV_LEFT",                       NAV_LEFT},
        {"NAV_RIGHT",                      NAV_RIGHT},
        {"REWIND",                         REWIND},
        {"FORWARD",                        FORWARD},
        {"NUM_1",                          NUM_1},
        {"NUM_2",                          NUM_2},
        {"NUM_3",                          NUM_3},
        {"NUM_4",                          NUM_4},
        {"NUM_5",                          NUM_5},
        {"NUM_6",                          NUM_6},
        {"NUM_7",                          NUM_7},
        {"NUM_8",                          NUM_8},
```

```
        {"NUM_9",                       NUM_9},
        {"NUM_0",                       NUM_0},
        {"NUM_TEN_PLUS",                NUM_TEN_PLUS},


        {"POWER",                       POWER},
        {NULL, 0},
};


static int parseFlags(char *val)
{
    const struct flaglist *flaglptr;
        int flags=0;
        char *flag,*help;


        flag=help=val;


        while(flag!=NULL)
        {
                while(*help!='|' && *help!=0) help++;
                if(*help=='|')
                {
                        *help=0;help++;
                }
                else
                {
                        help=NULL;
                }
                flaglptr=all_flags;
                while(flaglptr->name!=NULL){
                        if(strcmp(flaglptr->name,flag)==0){
                                flags=flags|flaglptr->flag;
                                break;
                        }
                        flaglptr++;
                }
                if(flaglptr->name==NULL)
                {
                        return(0);
                }
                flag=help;
        }

       return(flags);
}


unsigned char ir_lookupButton(const char *buttonName)
{
    const struct flaglist *flaglptr;
        unsigned char command;

        command = 255;
        flaglptr=allCommands;
        while(flaglptr->name!=NULL){
                if(strnocasecmp(flaglptr->name, buttonName)==0){
                        command= flaglptr->flag;
                        break;
                }
                flaglptr++;
        }
        return command;
}
```

```
static void defineButton(struct ir_remote*remote, struct ir_ncode *codes, const char *commandI
{
        char temp[24];
        unsigned char command;


        command = ir_lookupButton(commandName);

        if (command == 255) {
                debugHi(("Bad Button: %s", commandName));
                return;
        }
        ir_initWords(command);
        fsd_getAttribute(buttonNode, "value", temp, 24);
        ir_code_init(&codes->code);
        ir_strtocode(temp, 1, (char)remote->bits, &codes->code);
        send(codes, remote, (unsigned short)remote->min_repeat);
        ir_endWords(command);
}

static void defineRemote(char * key, NodeId ruleNode, struct ir_remote *rem)
{
        char temp[24];

        if ((strnocasecmp("bits",key))==0){
                fsd_getAttribute(ruleNode, "value", temp, 24);
                rem->bits=atoi(temp);
        }

        else if (strnocasecmp("flags",key)==0){
                fsd_getAttribute(ruleNode, "value", temp, 24);
                rem->flags|=parseFlags(temp);
        }









        else if (strnocasecmp("header",key)==0){
                fsd_getAttribute(ruleNode, "pulse", temp, 24);
                rem->phead=atoi(temp);
                fsd_getAttribute(ruleNode, "space", temp, 24);
                rem->shead=atoi(temp);
        }

        else if (strnocasecmp("one",key)==0){
                fsd_getAttribute(ruleNode, "pulse", temp, 24);
                rem->pone=atoi(temp);
                fsd_getAttribute(ruleNode, "space", temp, 24);
                rem->sone=atoi(temp);
        }

        else if (strnocasecmp("zero",key)==0){
                fsd_getAttribute(ruleNode, "pulse", temp, 24);
                rem->pzero=atoi(temp);
                fsd_getAttribute(ruleNode, "space", temp, 24);
                rem->szero=atoi(temp);
        }


        else if (strnocasecmp("plead",key)==0){
                fsd_getAttribute(ruleNode, "value", temp, 24);
```

```
        rem->plead=atoi(temp);
}


else if (strnocasecmp("ptrail",key)==0){
        fsd_getAttribute(ruleNode, "value", temp, 24);
        rem->ptrail=atoi(temp);
}


else if (strnocasecmp("foot",key)==0){
        fsd_getAttribute(ruleNode, "pulse", temp, 24);
        rem->pfoot=atoi(temp);
        fsd_getAttribute(ruleNode, "space", temp, 24);
        rem->sfoot=atoi(temp);
}



else if (strnocasecmp("repeat",key)==0){
        fsd_getAttribute(ruleNode, "prepeat", temp, 24);
        rem->prepeat=atoi(temp);
        fsd_getAttribute(ruleNode, "srepeat", temp, 24);
        rem->srepeat=atoi(temp);
}


else if (strnocasecmp("pre_data_bits",key)==0){
        fsd_getAttribute(ruleNode, "value", temp, 24);
        rem->pre_data_bits=atoi(temp);
}


else if (strnocasecmp("pre_data",key)==0){
        fsd_getAttribute(ruleNode, "value", temp, 24);
        ir_strtocode(temp, 1, (char)rem->pre_data_bits, &rem->pre_data);
}


else if (strnocasecmp("post_data_bits",key)==0){
        fsd_getAttribute(ruleNode, "value", temp, 24);
        rem->post_data_bits=atoi(temp);
}


else if (strnocasecmp("post_data",key)==0){
        fsd_getAttribute(ruleNode, "value", temp, 24);
        ir_strtocode(temp, 1, (char)rem->post_data_bits, &rem->post_data);
}


else if (strnocasecmp("pre",key)==0){
        fsd_getAttribute(ruleNode, "ppre", temp, 24);
        rem->pre_p=atoi(temp);
        fsd_getAttribute(ruleNode, "spre", temp, 24);
        rem->pre_s=atoi(temp);
}


else if (strnocasecmp("post",key)==0){
        fsd_getAttribute(ruleNode, "ppost", temp, 24);
        rem->post_p=atoi(temp);
        fsd_getAttribute(ruleNode, "spost", temp, 24);
        rem->post_s=atoi(temp);
}
```

```
        else if (strnocasecmp("gap",key)==0){
                fsd_getAttribute(ruleNode, "value", temp, 24);
                rem->gap=atol(temp);
        }


        else if (strnocasecmp("repeat_gap",key)==0){
                fsd_getAttribute(ruleNode, "value", temp, 24);
                rem->repeat_gap=atol(temp);
        }


        else if (strnocasecmp("toggle_bit",key)==0){
                fsd_getAttribute(ruleNode, "value", temp, 24);
        rem->toggle_bit=atoi(temp);
        }


        else if (strnocasecmp("min_repeat",key)==0){
                fsd_getAttribute(ruleNode, "value", temp, 24);
        rem->min_repeat=atoi(temp);
        }


        else if (strnocasecmp("frequency",key)==0){
                fsd_getAttribute(ruleNode, "value", temp, 24);
        rem->freq=atoi(temp);
        }


        else if (strnocasecmp("duty_cycle",key)==0){
                fsd_getAttribute(ruleNode, "value", temp, 24);
        rem->duty_cycle=atoi(temp);

        }else{
                debugPutstrHi(("Error config:"));
                debugPutstrHi((key));
        }
}


void ir_configIrCodesRom(void)
{
        struct eprom_script_def script;
        short beginRombuffer, numBytes;

        if (epromGetScript(IRDATA, -1, &script) != -1) {
                beginRombuffer = script.location;
                numBytes = script.len;
                ir_initPointersFromRom(beginRombuffer, numBytes);
                debugHi(("irRom = %d %d", beginRombuffer, numBytes));
        }
        else {
                debugPutstrHi(("find IRDATA in eprom failed"));
        }

}

void ir_configIrCodes(void)
{

        struct ir_ncode codes;
```

```
        NodeId parentNode;
    NodeId ruleNode;
        NodeId buttonNode;
        char temp[24];
        struct eprom_script_def epromScript;
        short beginRomBuffer, thisRomBuffer;
        short numBytes;



        memset((char *)&remote, 0, sizeof(remote));
        beginRomBuffer = irdataOffset;


        parentNode = fsdint_findButton(NODE_ROOT, "rules", NULL);

    debugPutstrHi(("compile rules"));
        ruleNode = fsd_fetchNodeId(parentNode,FIRSTCHILD);
    while (!(ruleNode == NODE_EMPTY || ruleNode == NODE_ERROR) ) {
                fsd_getNodeName(ruleNode, temp, 24);
                defineRemote(temp, ruleNode,  &remote);
                debugHi(("node %s", temp));
        ruleNode = fsd_fetchNodeId(ruleNode, NEXTNODE);
    }


        debugPutstrHi(("compile buttons"));
        parentNode = fsdint_findButton(NODE_ROOT, "buttons", NULL);

    buttonNode = fsd_fetchNodeId(parentNode,FIRSTCHILD);
    while (!(buttonNode == NODE_EMPTY || buttonNode == NODE_ERROR) ) {
                fsd_getNodeName(buttonNode, temp, 24);
                defineButton(&remote, &codes, temp, buttonNode);
                debugHi(("node %s", temp));
        buttonNode = fsd_fetchNodeId(buttonNode, NEXTNODE);
    }

        numBytes = irdataOffset - beginRomBuffer - 1;
        thisRomBuffer = numScriptBuffers;
        debug(("IRDATA Script %d %d", beginRomBuffer, numBytes ));
        scriptBuffer[thisRomBuffer] = beginRomBuffer;
        numScriptBuffers++;


        epromScript.id = -1;
        epromScript.location = beginRomBuffer;
        epromScript.type = IRDATA;
        epromScript.len = numBytes;
        epromWriteScriptNumber(thisRomBuffer, &epromScript);
#ifndef PIC

        pc_writeFlash(beginRomBuffer, numBytes);
#endif
}


static void defineRemoteTest(struct ir_remote *rem)
{
        char temp[32];


        rem->bits=16;


        rem->flags = SPACE_ENC | REVERSE;
```

```
            rem->phead=8800;
            rem->shead=4400;


            rem->pone=550;
            rem->sone=1650;


            rem->pzero=550;
            rem->szero=550;


            rem->plead=0;


            rem->ptrail=550;


            rem->pfoot=0;
            rem->sfoot=0;



            rem->prepeat=8800;
            rem->srepeat=2200;



            rem->pre_data_bits=16;


            strcpy(temp, "0xCD72");
            ir_strtocode(temp, 1, (char)rem->pre_data_bits, &rem->pre_data);



            rem->post_data_bits=0;


            strcpy(temp, "");
            ir_strtocode(temp, 1, (char)rem->post_data_bits, &rem->post_data);


            rem->pre_p=0;
            rem->pre_s=0;


            rem->post_p=0;
            rem->post_s=0;


            rem->gap=38500;



            rem->repeat_gap=0L;


rem->toggle_bit=0;


rem->min_repeat=0;
```

```
        rem->freq=0;


        rem->duty_cycle=0;
}

static void defineButtonTest(struct ir_remote*remote, struct ir_ncode *codes, const char *com
{
        char temp[24];
        unsigned char command;

        debug(("TestButton: %s %s", commandName, value));
        strcpy(temp, value);
        command = ir_lookupButton(commandName);

        if (command == 255) {
                debugHi(("Bad Button: %s", commandName));
                return;
        }
        ir_initWords(command);
        ir_code_init(&codes->code);
        ir_strtocode(temp, 1, (char)remote->bits, &codes->code);
        send(codes, remote, (unsigned short)remote->min_repeat);
        ir_endWords(command);
}


void ir_configTest(void)
{
        struct ir_ncode codes;


        memset((char *)&remote, 0, sizeof(remote));

    debugPutstrHi(("compile Test rules"));
        defineRemoteTest(&remote);


        debugPutstrHi(("compile Test buttons"));

        defineButtonTest(&remote, &codes, "PLAY", "0xE718");
        defineButtonTest(&remote, &codes, "STOP", "0xE619");
}


#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#ifdef IR_RULES

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "fsdtablelarge.h"
#include "sendirrules.h"
#include "sendircommon.h"

#ifdef PIC
#include "tablereadwrite.h"
#include "delay.h"
#else
#include "pcromchip.h"
#endif


struct ir_remote *repeat_remote=NULL;
struct ir_ncode *repeat_code=NULL;



extern const unsigned char *flashMemory;
extern short      offsetFlashMemory;
extern short      irScriptBuffer;
extern short      currentScriptBuffer;
extern short      numScriptBuffers;
extern unsigned short      scriptBuffer[];


#define MAXIRDATA 10000
unsigned short irPointers[MAXIRCOMMAND];
short currIrCommandLength;



static short irmacros[MAXIRMACRO+1];



void ir_rulesInit(void)
{
        short i;


        for (i=0; i < MAXIRMACRO+1; i++) {
                irmacros[i] = NODE_ERROR;
        }
        for (i=0; i < MAXIRCOMMAND; i++) {
                irPointers[i] = -1;
        }
}
```

```
NodeId ir_findMacro(short butNumber, const char *butName)
{
        NodeId butLoc;

        fsd_switchRomBuffer(irScriptBuffer);
        if (butNumber >= 0 && butNumber <= MAXIRMACRO) {
                if ( irmacros[butNumber] == NODE_ERROR) {
                        butLoc = fsdint_findButton(NODE_ROOT, "IrMacro", butName);
                        irmacros[butNumber] = butLoc;
                }
                else {
                        butLoc = irmacros[butNumber];
                }
        }
        if (butLoc != NODE_ERROR) {
                butLoc = fsdint_formBufferNode(butLoc);
        }
        fsd_unswitchRomBuffer();
        return butLoc;
}



void ir_initWords(unsigned char command)
{
        if (command > MAXIRCOMMAND - 1) return;
        debug(("Command: %d at %d", command, irdataOffset));
        currIrCommandLength = 0;
        ir_addWord(0, command);
        ir_addWord(0, 0);
        irPointers[command] = irdataOffset;
}



void ir_addWord(char flag, unsigned long word)
{
        unsigned short newWord;
        do {
                if (word > 0x7ffe) {
                        newWord = 0x7ffe;
                }
                else {
                        newWord = (unsigned short)word;
                }
                word -= newWord;
                if (flag) newWord |= 0x8000;
                if (irdataOffset < MAXIRDATA) {
#ifdef PIC
                        TableWrite((unsigned char *)&flashMemory[irdataOffset], (unsigned c
#else
                        memcpy((unsigned char *)&flashMemory[irdataOffset], &newWord, sizec
#endif
                }
                irdataOffset += sizeof(unsigned short);
                currIrCommandLength += sizeof(unsigned short);

        } while (word > 0);
}


void ir_endWords(unsigned char command)
{
```

```
        unsigned short newWord;
        short offset;

        offset = irPointers[command];
        offset -= sizeof(unsigned short);

        newWord = currIrCommandLength - (2 * sizeof(unsigned short));
        if (irdataOffset < MAXIRDATA) {
#ifdef PIC
                TableWrite((unsigned char *)&flashMemory[offset], (unsigned char *)&newWord,
#else
                memcpy((unsigned char *)&flashMemory[offset], &newWord, sizeof(unsigned shor
#endif
        }
}


void ir_sendWords(unsigned char command)
{
        short address;
        unsigned short word;
        short length, i;

        if (command > MAXIRCOMMAND - 1) return;

        address = irPointers[command];
        if (address == -1) return;

        memcpy(&length, flashMemory+(long)address-sizeof(unsigned short), 2);
#ifdef PIC
        di();
#endif
        for (i = 0; i < length; i+=2) {
                memcpy(&word, flashMemory+(long)address, 2);
                if (word == 0xffff) break;

                if (word & 0x8000) {
                        ir_LedOn((unsigned short)(word & 0x7fff));
                }
                else {
                        ir_LedOff(word);
                }
                address += 2;
        }
#ifdef PIC
        ei();
#endif
}


void ir_initPointersFromRom(short address, short len)
{
        short offset = address;
        short command, length;

#ifndef PIC

                pc_readFlash(offset, len);
#endif

        while(offset < address + len) {
                memcpy(&command, &flashMemory[offset], 2);
                offset += 2;
                memcpy(&length, &flashMemory[offset], 2);
                offset += 2;
                irPointers[command] = offset;
                offset += length;
```

```
        }
        offsetFlashMemory = address;
        currentScriptBuffer = numScriptBuffers;
        numScriptBuffers++;

        scriptBuffer[currentScriptBuffer] = offsetFlashMemory;
        offsetFlashMemory += len;
}


void ir_sendNumbersString(const char *sNum)
{
    short i, len;
    char sNumber;

        len = strlen(sNum);
    for (i = 0; i < len; i++ ) {
        sNumber = *sNum++;
                  sNumber -= '0';
        switch (sNumber) {
            case 0:
                                ir_sendWords(NUM_0);
                                break;
            case 1:
                ir_sendWords(NUM_1);
                                break;
            case 2:
                ir_sendWords(NUM_2);
                                break;
            case 3:
                ir_sendWords(NUM_3);
                                break;
            case 4:
                ir_sendWords(NUM_4);
                                break;
            case 5:
                ir_sendWords(NUM_5);
                                break;
            case 6:
                ir_sendWords(NUM_6);
                                break;
            case 7:
                ir_sendWords(NUM_7);
                                break;
            case 8:
                ir_sendWords(NUM_8);
                                break;
            case 9:
                ir_sendWords(NUM_9);
                                break;
        }
#ifdef PIC

#endif
    }
}


static char is_biphase(struct ir_remote *remote)
{
        if(remote && (remote->flags&RC5_CODE || remote->flags&RC6_CODE)) return(1);
        else return(0);
}
```

```
static char is_rc6(struct ir_remote *remote)
{
        if(remote && remote->flags&RC6_CODE) return(1);
        else return(0);
}


static char is_rcmm(struct ir_remote *remote)
{
        if(remote && remote->flags&RCMM) return(1);
        else return(0);
}



static char is_raw(struct ir_remote *remote)
{
         if(remote && remote->flags&RAW_CODES) return(1);
         else return(0);
}




static char is_const(struct ir_remote *remote)
{
         if(remote && remote->flags & CONST_LENGTH) return(1);
         else return(0);
}



static char has_header(struct ir_remote *remote)
{
         if(remote && remote->phead>0 && remote->shead>0) return(1);
         else return(0);
}

static char has_foot(struct ir_remote *remote)
{
         if(remote && remote->pfoot>0 && remote->sfoot>0) return(1);
         else return(0);
}




static char has_repeat(struct ir_remote *remote)
{
         if(remote && remote->prepeat>0 && remote->srepeat>0) return(1);
         else return(0);
}

static char has_repeat_gap(struct ir_remote *remote)
{
         if(remote && remote->repeat_gap>0) return(1);
         else return(0);
}
```

```
static char has_pre(struct ir_remote *remote)
{
        if(remote && remote->pre_data_bits>0) return(1);
        else return(0);
}

static char has_post(struct ir_remote *remote)
{
        if(remote && remote->post_data_bits>0) return(1);
        else return(0);
}
```

```
unsigned long s_strtoul(char *val, char **endptr, char base)
{
        unsigned long result=0;
        unsigned char c;

        while(*val=='\t' || *val==' ') val++;
        if(base==0)
                if(val[0]=='0')
                        if(val[1]=='x' || val[1]=='X')
                        {
                                base=16;
                                val+=2;
                        }
                        else
                        {
                                val++;
                                base=8;
                        }
                else
                        base=10;


        while(1)
        {
                c = *val;
                if(c >= '0' && c <= '9') c = c - '0';
                else if(c >= 'a' && c <= 'f') c = (c - 'a') + 10;
                else if(c >= 'A' && c <= 'F') c = (c - 'A') + 10;
                else break;

                result *= base;
                result += c;
                val++;
        }
```

```
        *endptr=val;
        return result;
}

void send_space(unsigned long length)
{









        ir_addWord(0, length);
}

void send_pulse(unsigned long length)
{

        ir_addWord(1, length);









}



void ir_strtocode(char *val, char which, char numBits, ir_code *code)
{
        unsigned long value;
        char *endptr;

        value = s_strtoul(val,&endptr,0);
        if(strlen(endptr)!=0 || strlen(val)==0)
        {
                code->data[which] = 0;
                code->bits[which] = 0;
                return;
        }
        code->data[which] = value;
        code->bits[which] = numBits;
        return;
}


void ir_code_init(ir_code *code)
{
        char i;
        for (i=0; i < IR_CODE_LENGTH; i++) {
                code->data[i] = 0;
                code->bits[i] = 0;
        }
}

static char ir_code_hasData(ir_code *code)
{
        char i;
        for (i=0; i < IR_CODE_LENGTH; i++) {
                if (!(code->bits[i] == 0)) return 1;
        }
        return 0;
```

```
}



void ir_send_data_long(unsigned long value, char bits)
{
        while(bits-- > 0) {
                if (value & 1) {

                }
                else {

                }
                value = value >> 1;
        }
}


void ir_set_bit(ir_code *code, short bitnum, char data)
{
        short which=IR_CODE_LENGTH-1;
        short whichBit=bitnum;
        char totalBits=0;

        if ((short)bitnum < 0) return;

        for (which=IR_CODE_LENGTH-1; which >=0; which--) {
                totalBits += code->bits[which];
                if (bitnum < totalBits ) {


                        code->data[which] &= ~(1 << whichBit);
                        code->data[which] |= (data ? 1:0) << whichBit;
                        break;
                }
                whichBit -= code->bits[which];
        }
}


char ir_get_bit(ir_code *code, short bitnum)
{
        short which=IR_CODE_LENGTH-1;
        short whichBit=bitnum;
        char totalBits=0;

        if (bitnum < 0) return 0;

        for (which=IR_CODE_LENGTH-1; which >=0; which--) {
                totalBits += code->bits[which];
                if (bitnum < totalBits ) {

                        if (code->data[which] & (1 << whichBit) ) {
                                return 1;
                        }
                        else {
                                return 0;
                        }
                }
                whichBit -= code->bits[which];
        }
        return 0;
}
```

```
void ir_reverse(ir_code *inCode, ir_code *outCode)
{
        char i, sourceBit, bitnum;
        char destBit;
        char totalBits=0;

        ir_code_init(outCode);

        for(i=0; i < IR_CODE_LENGTH; i++) {
                totalBits += inCode->bits[i];
                outCode->bits[i] = inCode->bits[i];
        }

        destBit = totalBits-1;
        for(sourceBit=0; sourceBit < totalBits; sourceBit++)
        {
                bitnum = ir_get_bit(inCode, sourceBit);

                if (bitnum)  {
                        ir_set_bit(outCode, destBit, bitnum);
                }

                destBit--;
        }

}


void send_header(struct ir_remote *remote)
{
        if(has_header(remote))
        {

                send_pulse(remote->phead);
                send_space(remote->shead);
        }
}


void send_foot(struct ir_remote *remote)
{
        if(has_foot(remote))
        {

                send_space(remote->sfoot);
                send_pulse(remote->pfoot);
        }
}


void send_lead(struct ir_remote *remote)
{
        if(remote->plead!=0)
        {

                send_pulse(remote->plead);
        }
}


void send_trail(struct ir_remote *remote)
{
        if(remote->ptrail!=0)
        {
```

```
                send_pulse(remote->ptrail);
        }
}

void send_data(struct ir_remote *remote, ir_code *inData, int bits)
{
        char i;
        ir_code data;


        if(!(remote->flags&REVERSE)) {
                ir_reverse(inData, &data);
        }
        else {
                memcpy(&data, inData, sizeof(data));

        }
        for(i=0;i<bits;i++)
        {
                if(ir_get_bit(&data, i))
                {

                        if(is_biphase(remote))
                        {
                                if(is_rc6(remote) && i+1==remote->toggle_bit)
                                {
                                        send_space(2*remote->sone);
                                        send_pulse(2*remote->pone);
                                }
                                else
                                {
                                        send_space(remote->sone);
                                        send_pulse(remote->pone);
                                }
                        }
                        else
                        {
                                send_pulse(remote->pone);
                                send_space(remote->sone);
                        }
                }
                else
                {

                        if(is_rc6(remote) && i+1==remote->toggle_bit)
                        {
                                send_pulse(2*remote->pzero);
                                send_space(2*remote->szero);
                        }
                        else
                        {
                                send_pulse(remote->pzero);
                                send_space(remote->szero);
                        }
                }
        }
}

void send_pre(struct ir_remote *remote)
{
        ir_code pre;

        if(has_pre(remote))
        {


                memcpy(&pre, &remote->pre_data, sizeof(pre));
```

```
                if(remote->toggle_bit>0)
                {
                        if(remote->toggle_bit<=remote->pre_data_bits)
                        {
                                ir_set_bit(&pre,
                                        (char)(remote->pre_data_bits - remote->toggle_bit
                                        (char)remote->repeat_state);
                        }
                }

                if (ir_code_hasData(&pre)) {
                        send_data(remote, &pre, remote->pre_data_bits);
                }
                if(remote->pre_p>0 && remote->pre_s>0)
                {
                        send_pulse(remote->pre_p);
                        send_space(remote->pre_s);
                }
        }
}

void send_post(struct ir_remote *remote)
{
        if(has_post(remote))
        {
                ir_code post;



                memcpy(&post, &remote->post_data, sizeof(post));
                if(remote->toggle_bit>0)
                {
                        if(remote->toggle_bit>remote->pre_data_bits
                           +remote->bits
                           &&
                           remote->toggle_bit<=remote->pre_data_bits
                           +remote->bits
                           +remote->post_data_bits)
                        {
                                ir_set_bit(&post,
                                        (char)(remote->pre_data_bits + remote->bits
                                                                + remote->post_data_bi
                                        (char)remote->repeat_state);
                        }
                }

                if(remote->post_p>0 && remote->post_s>0)
                {
                        send_pulse(remote->post_p);
                        send_space(remote->post_s);
                }
                if (ir_code_hasData(&post)) {
                        send_data(remote, &post, remote->post_data_bits);
                }
        }
}



void send_repeat(struct ir_remote *remote)
{

        send_lead(remote);
        send_pulse(remote->prepeat);
        send_space(remote->srepeat);
        send_trail(remote);
}
```

```
void send_code(struct ir_remote *remote, ir_code *code)
{
        if(remote->toggle_bit>0)
        {
                if(remote->toggle_bit>remote->pre_data_bits
                   &&
                   remote->toggle_bit<=remote->pre_data_bits
                   +remote->bits)
                {
                        ir_set_bit(code,
                                (char)(remote->pre_data_bits        + remote->bits - remot
                                (char)remote->repeat_state);
                }
                else if(remote->toggle_bit>remote->pre_data_bits
                        +remote->bits
                        +remote->post_data_bits)
                {

                }
        }


        if(repeat_remote==NULL || !(remote->flags&NO_HEAD_REP))
                send_header(remote);


        send_lead(remote);


        send_pre(remote);
        send_data(remote,code,remote->bits);
        send_post(remote);
        send_trail(remote);
        if(repeat_remote==NULL || !(remote->flags&NO_FOOT_REP))
                send_foot(remote);
}

int init_send(struct ir_remote *remote,struct ir_ncode *code)
{

        if(is_rcmm(remote))
        {

                return(0);
        }


        if(repeat_remote != NULL && has_repeat(remote))
        {


                if(remote->flags & REPEAT_HEADER && has_header(remote))
                {
                        send_header(remote);
                }
                send_repeat(remote);
        }
        else
        {
                if(!is_raw(remote))
                {
                        send_code(remote,&code->code);
                }
                else
                {
```

```
                        sendRaw(code->signals,code->length);
                }
        }

        if(is_const(remote))
        {
                remote->remaining_gap=remote->gap;
        }
        else
        {
                if(has_repeat_gap(remote) &&
                   repeat_remote!=NULL &&
                   has_repeat(remote))
                {
                        remote->remaining_gap=remote->repeat_gap;
                }
                else
                {
                        remote->remaining_gap=remote->gap;
                }
        }
        return(1);
}


void sendRaw(unsigned long *raw, int cnt)
{
        int i;



    for (i=0;i<cnt;i++) {
        if (i%2) send_space(raw[i]);
        else send_pulse(raw[i]);
    }

}
```

```
void send (struct ir_ncode *data, struct ir_remote *remote, unsigned short reps)
{
    if (!remote) return;


        if(remote->toggle_bit > 0) {
```

```
                remote->repeat_state = !remote->repeat_state;
        }

        init_send(remote,data);
    send_space(remote->remaining_gap);
        if (reps>0)
        {
                repeat_remote=remote;
                repeat_code=data;
                for (; reps > 0; --reps)
                {
                        init_send(remote,data);
                        send_space(remote->remaining_gap);
                }
                repeat_remote=NULL;
                repeat_code=NULL;
        }
}
```

```
#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef __DELAY_H
#define __DELAY_H


#include "config.h"

extern unsigned char delayus_variable;

#if PIC_CLK == 4000000
        #define DelayDivisor 4
        #define WaitFor1Us asm("nop")
        #define Jumpback asm("goto $ - 4")
#elif PIC_CLK == 8000000
        #define DelayDivisor 2
        #define WaitFor1Us asm("nop")
        #define Jumpback asm("goto $ - 4")
#elif PIC_CLK == 10000000
        #define DelayDivisor 2
        #define WaitFor1Us asm("nop"); asm("nop");
        #define Jumpback asm("goto $ - 6")
#elif PIC_CLK == 16000000
        #define DelayDivisor 1
        #define WaitFor1Us asm("nop")
        #define Jumpback asm("goto $ - 4")
#elif PIC_CLK == 20000000
        #define DelayDivisor 1
        #define WaitFor1Us asm("nop"); asm("nop");
        #define Jumpback asm("goto $ - 6")
#elif PIC_CLK == 32000000
        #define DelayDivisor 1
        #define WaitFor1Us asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop")
        #define Jumpback asm("goto $ - 12")
#else
        #error delay.h - please define PIC_CLK correctly
#endif

#define DelayUs(x) { \
                        delayus_variable=(unsigned char)(x/DelayDivisor); \
                        asm("movlb (_delayus_variable) >> 8"); \
                        WaitFor1Us; } \
                        asm("decfsz (_delayus_variable)&0ffh,f"); \
                        Jumpback;
```

```
#define LOOP_CYCLES_CHAR   9
#define timeout_char_us(x) (long)(((x)/LOOP_CYCLES_CHAR)*(PIC_CLK/1000000/4))

#define LOOP_CYCLES_INT             16
#define timeout_int_us(x)  (long)(((x)/LOOP_CYCLES_INT)*(PIC_CLK/1000000/4))


#define timeout_int_lobyte_zero_us(x)      (long)(((x)/LOOP_CYCLES_INT)*(PIC_CLK/4.0)&0xFF00


void DelayBigUs(unsigned int cnt);
void DelayMs(unsigned int cnt);
void DelayS(unsigned char cnt);

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#include "fsdtablelarge.h"
#include "fsdinterpretertable.h"
#include <string.h>
#include  <time.h>
#ifdef IR_RULES
#include "sendircommon.h"
#endif
#ifdef IR_UNIV_CHIP
#include "sendirunivchip.h"
#endif

#ifdef DEBUG
#include <stdio.h>
#endif

extern char TEMPBUFFER[];
#ifdef PIC
#include <pic18.h>
extern long TICKS;
void checkButtons(void);
#endif


static BOOL condition(const char *name, const char *oper, const char *value);
static BOOL testCondition(NodeId commandNode);
static NodeId setupTrick(const NodeId commandNode);


void PushPlayInitialize(void);
static   const char        **sCommands;
static   short (*processCommand)(short iCommand, NodeId commandNode, NodeId buttons[], short
static   void     (*infoCaller)(const char *msg);
static  BOOL      bStopInterpreter;
static  BOOL      bInterpreterStopped;
static  BOOL      bStopExecuteButton;
static   BOOL     bStopExecuteButtonInternal;

extern   short    maxNode;
extern   short    maxAttribute;
extern   TextLoc  maxTextLoc;
extern  short     currentScriptBuffer;



NodeId fsdint_formBufferNode(NodeId inNode)
{
        NodeId ret;

        if (inNode == NODE_ERROR) return inNode;
        if (inNode < 0)
                ret = ((currentScriptBuffer & 7) << 12) + 0x8000;
        else
                ret = currentScriptBuffer << 12;

        ret += inNode & 0xFFF;
        return ret;
```

```
}

NodeId fsdint_getBufferNode(NodeId inNode)
{
        if (inNode == NODE_ERROR) return inNode;
        if (inNode & 0x8000 ) {
                return (inNode & 0xFFF) + 0xF000;
        }
        else {
                currentScriptBuffer = (inNode & 0x7000) >> 12;
                return inNode & 0xFFF;
        }
}

void fsdint_initCommands(const char *Commands[], short (*procCall) (short, NodeId, NodeId[], s
{

        fsd_setMainScriptBuffer();

        sCommands = Commands;
        processCommand = procCall;
        infoCaller = infoCall;
        bStopInterpreter = FALSE;
        bInterpreterStopped = TRUE;
        fsdint_ButtonsOn();
}

short fsdint_lookupCommand(const char *command)
{
        short cnt;
        cnt = 0;
        while (1) {
                if (sCommands[cnt] == NULL) break;
                if (strcmp(sCommands[cnt], command) == 0) {
                        return cnt;
                }
                cnt++;
        }
        return -1;
}

void fsdint_ButtonsOffInternal(void)
{
    bStopExecuteButtonInternal = TRUE;
}

void fsdint_ButtonsOnInternal(void)
{
    bStopExecuteButtonInternal = FALSE;
}

void fsdint_ButtonsOff(void)
{
    bStopExecuteButton = TRUE;
}

void fsdint_ButtonsOn(void)
{
    bStopExecuteButton = FALSE;
}
```

```
void fsdint_executeButton(const char *sName)
{
    if (!(bStopExecuteButton || bStopExecuteButtonInternal)) {
        SEnqueue(sName);
    }
}

void  fsdint_startInterpreter()
{
    NodeId buttonNode;
         TextLoc loc;
         PtrTextLoc pCommand;
         long start;
         short firstLoop;
#ifdef AUTORUN
         int  count=0;
#endif
    NodeId globalNode;
         short ret;

         start = 0;
         firstLoop = TRUE;
         bStopInterpreter = FALSE;
         pCommand = NULL;
    bInterpreterStopped = FALSE;
    EmptyIStack();

    globalNode = fsd_slotNode();
    IPush(fsdint_formBufferNode(globalNode));
    fsdint_ButtonsOn();
    debugPutstrHi(("Intrp Started"));

         while (TRUE) {
#ifdef   PIC
                  checkButtons();
#endif
#ifdef AUTORUN

                  if (count == 0 ) {
                           fsdint_executeButton("Startup");
                           count++;
                  }
                  else if (count == 1) {
                           fsdint_executeButton("Button0");
                           count++;
                  }

#endif
                  if (!pCommand) {
                           loc = fsd_slotTextBlock();
                           pCommand = fsd_fetchTextLocPtr(loc);
                  }
                  if (pCommand == NULL) {
                           debugPutstrHi("no TextLoc!");
                           return;
                  };

                  if (firstLoop) {
                           fsdint_executeButton("Startup");
                  }
#ifdef MANUALINPUT
                  if (!firstLoop && bStopInterpreter == FALSE) {

                           puts("Button?: ");
```

```
                        gets(pCommand);
                        if (strlen(pCommand)) {
                                fsdint_executeButton(pCommand);
                        }
                }

#endif
                firstLoop = FALSE;
                start = GetTicks();

                if (bStopInterpreter == TRUE ) {
                        fsdint_ButtonsOff();
                        debugPutstrHi("Intrp Stopped");
                        bInterpreterStopped = TRUE;
                        break;
                }
                ret = SDequeue(pCommand, MAX_COMMANDSIZE);
                if (ret) {
                        fsd_setMainScriptBuffer();
                        EmptyRStack();

                        while (ICount() > 1) {
                                IPop();
                        }
                        buttonNode = fsdint_findButton(NODE_ROOT, "Button", pCommand);
                        if (buttonNode == NODE_ERROR ) {
                                debugHi(("No script %s", pCommand));
                        } else {

                                debugHi(("Start: %s", pCommand));

                                buttonNode = fsdint_formBufferNode(buttonNode);
                                fsd_scratchTextBlock(loc);
                                pCommand = NULL;
                                fsdint_interpretButton (buttonNode);
#ifdef PIC
                                debugHi(("Time: %d", (short)((GetTicks() - start) / 1000))
#else
                                debugHi(("Time: %d", GetTicks() - start ));
#endif
                                debugHi(("max: %d %d %d", maxNode,maxAttribute,maxTextLoc)
                        }
                }
        }
        if (pCommand)
                fsd_scratchTextBlock(loc);
        fsd_scratchNode(globalNode);
    EmptyIStack();
    EmptyRStack();
    EmptySQueue();
}


void fsdint_Initialize(void)
{

        fsd_Initialize();

        ir_Initialize();

        fsd_LoadMainScript();

        PushPlayInitialize();
}


void fsdint_RunInterpreter(void) {
```

```
        while(1) {
                fsdint_Initialize();
                fsdint_startInterpreter();
        }
}


void fsdint_Restart(void)
{
        epromInitialize(TRUE);
        bStopInterpreter = TRUE;
#ifdef PIC
        asm("reset");
#endif

}


void fsdint_Reset(void)
{
        bStopInterpreter = TRUE;

        fsd_clearEpromScript(MAINSCRIPT, -1);
}


void fsdint_GetIrScript(void)
{

        epromInitialize(TRUE);
        bStopInterpreter = TRUE;
}



void fsdint_SetIrScript(short scriptId)
{
        epromInitialize(TRUE);
        epromWriteWord(EPROM_IR_SCRIPTID, scriptId);
        bStopInterpreter = TRUE;
}









NodeId fsdint_findButton(NodeId startNode,  const char *sName, const char *sId)
{
    NodeId id;
        NodeId root;
        NodeId retId;
        TextLoc loc;
        PtrTextLoc nodeName;

    retId = NODE_ERROR;
        if (startNode == NODE_ROOT) {
                root = fsd_getRootNode();
        } else {
```

```
                root = startNode;
        }
        if (root == NODE_ERROR) {
                return NODE_ERROR;
        }
        loc = fsd_slotTextBlock();
        nodeName = fsd_fetchTextLocPtr(loc);
        if (nodeName == (PtrTextLoc)NODE_ERROR) return NODE_ERROR;
    id = fsd_fetchNodeId(root, FIRSTCHILD);

    while (!(id == NODE_EMPTY || id == NODE_ERROR) ) {

                fsd_getNodeName(id, nodeName, CHAR_BUFFERSIZE);
        if ( strnocasecmp(nodeName,sName) == 0 ) {

                        if (sId == NULL) {
                                retId = id;
                                break;
                        }
                        fsd_getAttribute(id, "id", nodeName, CHAR_BUFFERSIZE);
            if ( strnocasecmp(nodeName,sId) == 0) {
                    retId = id;
                                        break;
            }
        }
             id = fsd_fetchNodeId(id,NEXTNODE);
    }
         fsd_scratchTextBlock(loc);
    return retId;
}


void fsd_getCommandParameter(const char *name, const NodeId commandNode, char *buffer, const
{

    IPush(fsdint_formBufferNode(commandNode));
    fsdint_fetch(name, buffer, len);
    IPop();
}

static NodeId setupTrick(const NodeId commandNode)
{
        char buffer[CHAR_BUFFERSIZE];
    NodeId trickNode;

        fsd_getCommandParameter("id", commandNode, buffer, CHAR_BUFFERSIZE);
        trickNode = fsdint_findButton(NODE_ROOT, "Trick", buffer);
        if ( trickNode == NODE_ERROR ) {
                debugHi(("No Trick: %s" , buffer));
                return NODE_ERROR;
        }
        else {
                debugHi(("Trick Start: %s", buffer));
                return trickNode;
        }
 }




void fsdint_interpretButton(const NodeId buttonNode)
{
    NodeId commandNode;
    NodeId trickNode;
    short iCmd;
```

```
        short pos, i;
        short count;
    NodeId  topNode, nextNode;
        NodeId buttons[NUMRETURNNODES];

        debugHi(("Stack Counts: %d %d" , ICount(), RCount()));

        topNode = fsdint_getBufferNode(buttonNode);
        pos = 0;
        while (1) {
#ifdef  PIC
                checkButtons();
#endif
                if (pos == 0) {

                    IPush(fsdint_formBufferNode(topNode));
                }
                commandNode = fsd_getChildByPos(topNode, pos);
                if (commandNode == NODE_ERROR)  {

                        IPop();

                        if (RCount() > 0) {
                                pos = RPop();
                                nextNode = RPop();

                                topNode = fsdint_getBufferNode(nextNode);
                                continue;
                        }
                        else {
                                break;
                        }
                }



        if ( !QueueIsEmpty()  || bStopInterpreter )  {
                        break;
                }

                fsd_getNodeName(commandNode, TEMPBUFFER, CHAR_BUFFERSIZE);

        iCmd = fsdint_lookupCommand(TEMPBUFFER);
        if ( iCmd != -1 ) {

            switch (iCmd) {



                                        case 27:
                                                if (testCondition(commandNode) ) {

                                                        RPush(fsdint_formBufferNode(top
                                                        RPush(++pos);
                                                        topNode = commandNode;
                                                        pos = 0;

                            continue;
                        }
                                                break;


                                        case 29:
                                                trickNode = setupTrick(commandNode);
                                                if (trickNode != NODE_ERROR) {
                                                        RPush(fsdint_formBufferNode(top
                                                        RPush(++pos);
```

```
                                                    topNode = trickNode;
                                                    pos = 0;
                                                    continue;
                        }
                                                break;

                default:

                    IPush(fsdint_formBufferNode(commandNode));
                                                count = processCommand(iCmd, commandNode
                                                if (count > 0) {




                                                    RPush(fsdint_formBufferNode(top
                                                    RPush(++pos);

                                                    RPush(fsdint_formBufferNode(com
                                                    RPush(1);
                                                    if (count > 1) {


                                                        for (i = count - 1; i :
                                                            RPush(buttons
                                                            RPush(0);
                                                        }
                                                    }

                                                    topNode = fsdint_getBufferNode(
                                                    pos = 0;
                                                    continue;
                                                }
                                                else {
                                                    IPop();
                                                }
            }
        }
                pos++;
        }
}
```

```
static BOOL testCondition(NodeId commandNode)
{
    char name[CHAR_BUFFERSIZE], value[CHAR_BUFFERSIZE], oper[CHAR_BUFFERSIZE];

    fsd_getCommandParameter("id", commandNode, name, CHAR_BUFFERSIZE);
    fsd_getCommandParameter("value", commandNode,value, CHAR_BUFFERSIZE);
    fsd_getCommandParameter("oper", commandNode, oper, CHAR_BUFFERSIZE);
    return  condition(name, oper, value);
}


static BOOL condition(const char *name, const char *oper, const char *value)
{

    char sValue[CHAR_BUFFERSIZE];
    short result;


        fsdint_fetch(name, sValue, CHAR_BUFFERSIZE);

    result = strcmp(sValue, value);

        if (strcmp(oper, "eq") == 0) {
                if (result == 0) return TRUE;
        } else if (strcmp(oper, "neq") == 0) {
                if (result != 0) return TRUE;
        } else if (strcmp(oper, "gt") == 0) {
                if (result == 1) return TRUE;
        } else if (strcmp(oper, "lt") == 0) {
                if (result == -1) return TRUE;
        }

        return FALSE;

}



void fsdint_store(const char *name, const char *value)
{
    NodeId node,nextNode;
        short saveRomBuffer;
```

```
        saveRomBuffer = currentScriptBuffer;


    nextNode = IPeek((short)(ICount() - 1));
        node = fsdint_getBufferNode(nextNode);
    if (node != NODE_ERROR ) {
        fsd_setAttribute (node, name, value);
    }
        currentScriptBuffer = saveRomBuffer;
}


void fsdint_fetch(const char *name, char *buffer, const short len)
{
    short iCnt;
    short i;
    NodeId node,nextNode;
        TextLoc loc1, loc2;
        PtrTextLoc sName, sAttrib;
        short saveRomBuffer;

        saveRomBuffer = currentScriptBuffer;
        loc1 = fsd_slotTextBlock();
        loc2 = fsd_slotTextBlock();
        sName = fsd_fetchTextLocPtr(loc1);
        sAttrib = fsd_fetchTextLocPtr(loc2);

        if (!sName || !sAttrib) {
                buffer[0] = 0;
                debugPutstrHi("slottextBlock failure");
                goto exit;
        }


        if (strlen(name) > (CHAR_BUFFERSIZE - 1)) {
                buffer[0] = 0;
                goto exit;
        }


        strcpy(sName, name);
        iCnt = ICount();
        i = 0;
        while(1) {
                nextNode = IPeek(i);
                node = fsdint_getBufferNode(nextNode);
                if ( node != NODE_ERROR ) {
                        fsd_getAttribute(node, sName, sAttrib, CHAR_BUFFERSIZE);
                        if (strlen(sAttrib) > 0) {

                                if (sAttrib[0] == '@' ) {

                                        strcpy(sName, &sAttrib[1]);
                                        i = 0;
                                        continue;
                                }
                                else {
                                        break;
                                }
                        }
                }
                i++;
                if (i >= iCnt) break;
        }
        if (strlen(sAttrib) >= (unsigned short)(len - 1) ) {
                sAttrib[len - 1] = 0;
```

```
        }
        strcpy(buffer, sAttrib);
exit:
        fsd_scratchTextBlock(loc1);
        fsd_scratchTextBlock(loc2);
        currentScriptBuffer = saveRomBuffer;
}



void fsdint_increment(const char *name, const short minValue, const short maxValue)
{
    short iValue;
        TextLoc loc1;
        PtrTextLoc sValue;

        loc1 = fsd_slotTextBlock();
        sValue = fsd_fetchTextLocPtr(loc1);

        if (sValue != (PtrTextLoc)NODE_ERROR) {
                fsdint_fetch(name, sValue, CHAR_BUFFERSIZE);
                iValue = fsd_getInteger(sValue);
                iValue++;
                if (iValue > maxValue) iValue = minValue;
                longToAscii(iValue, sValue);
                fsdint_store (name, sValue);
        }
        else {
                debugPutstrHi("slottextBlock failure");
        }

        fsd_scratchTextBlock(loc1);
}


void fsdint_append(const char *name, const char *value)
{
        TextLoc loc1;
        PtrTextLoc sValue;

        loc1 = fsd_slotTextBlock();
        sValue = fsd_fetchTextLocPtr(loc1);

        if (sValue != (PtrTextLoc)NODE_ERROR) {

                fsdint_fetch(name, sValue, CHAR_BUFFERSIZE);

                if ( (strlen(sValue) + strlen(value)) < CHAR_BUFFERSIZE ) {
                        strcat(sValue, value);
                }
                fsdint_store (name, sValue);
        }
        else {
                debugPutstrHi("slottextBlock failure");
        }

        fsd_scratchTextBlock(loc1);
}


long GetTicks(void)
{
#ifdef PIC
        return TICKS;
#else
```

```
        clock_t ticks;
        ticks = clock();
        return (long)ticks;
#endif
}


void  fsdint_delay(long seconds, long milliseconds)
{
        long ticks;
    long intDelay;

#ifndef PIC
        return;
#endif

        intDelay = 0;
        if (seconds > 0)  intDelay = seconds * 1000;
    intDelay += milliseconds;
    ticks = GetTicks() + intDelay;
    while (ticks > GetTicks()) {

#ifdef   PIC
                checkButtons();
#endif
        if ( !QueueIsEmpty()  || bStopInterpreter ) break;
        }
}


void  fsdint_hardDelay(long seconds, long milliseconds)
{
    long ticks;
    long intDelay;

#ifndef PIC
        return;
#endif

        intDelay = 0;
        if (seconds > 0)  intDelay = seconds * 1000;
    intDelay += milliseconds;
    ticks = GetTicks() + intDelay;
    while (ticks > GetTicks()) {
#ifdef   PIC
                checkButtons();
#endif
       if ( bStopInterpreter ) break;
        }
}
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef __squeue_h_
#define __squeue_h_

#include <string.h>



#define QUEUE_DIM 4
#define MAXQUEUELENGTH 16



void SEnqueue(const char *el);

char SDequeue(char *el, const int len);

void EmptySQueue(void);
char QueueIsEmpty(void);
char QueueIsFull(void);

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef _SERIAL_H_
#define _SERIAL_H_

#define BAUD 9600
#define FOSC PIC_CLK
#define NINE 0
#define OUTPUT 1
#define INPUT 1

#define SPBRG_DIVIDER ((int)(FOSC/(16UL * BAUD) -1))
#define HIGH_SPEED 1

#if NINE == 1
#define NINE_BITS 0x40
#else
#define NINE_BITS 0
#endif

#if HIGH_SPEED == 0
#define SPEED 0x4
#else
#define SPEED 0
#endif

void init_comms(void);
void putch(unsigned char);
unsigned char getch(void);
unsigned char getche(void);
char *getsNoEcho(char *s);
char *gets(char *s);
int puts(const char *s);


#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef _SERIAL_H_
#define _SERIAL_H_

#define BAUD 9600
#define FOSC PIC_CLK
#define NINE 0
#define OUTPUT 1
#define INPUT 1

#define SPBRG_DIVIDER ((int)(FOSC/(16UL * BAUD) -1))
#define HIGH_SPEED 1

#if NINE == 1
#define NINE_BITS 0x40
#else
#define NINE_BITS 0
#endif

#if HIGH_SPEED == 0
#define SPEED 0x4
#else
#define SPEED 0
#endif

void init_comms(void);
void putch(unsigned char);
unsigned char getch(void);
unsigned char getche(void);
char *getsNoEcho(char *s);
char *gets(char *s);
int puts(const char *s);


#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

void errorBeep(void);
void goodBeep(void);
void keypressBeep(void);

#ifdef PIC
void beep( int frequency, int duration );

#define BEEPER RC0




#define  c0       262
#define  cS0      277
#define  d0       294
#define  dS0      311
#define  e0       330
#define  f0       349
#define  fS0      370
#define  g0       392
#define  gS0      415
#define  a0       440
#define  aS0      466
#define  b0       494


#define  c1       523
#define  cS1      554
#define  d1       587
#define  dS1      622
#define  e1       659
#define  f1       698
#define  fS1      740
#define  g1       784
#define  gS1      831
#define  a1       880
#define  aS1      932
#define  b1       988


#define  c2       1047
#define  cS2      1109
#define  d2       1174
#define  dS2      1245
#define  e2       1319
#define  f2       1397
#define  fS2      1480
#define  g2       1568
#define  gS2      1661
#define  a2       1760
#define  aS2      1965
#define  b2       1976



#define  c3       2093
#define  cS3      2217
```

```
#define  d3        2344
#define  dS3       2489
#define  e3        2637
#define  f3        2794
#define  fS3       2960
#define  g3        3136
#define  gS3       3322
#define  a3        3520
#define  aS3       3729
#define  b3        3951

#define  SIXTEENTH 63
#define            EIGHTH  125
#define  QUARTER   250
#define            HALF  500
#define            WHOLE  1000

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef __i2c_ccs_H
#define __i2c_ccs_H


void random_write(char dev_adr, int mem_adr, char dat);
char random_read(char dev_adr, int mem_adr);
void random_readM(char dev_adr, int mem_adr, void *Data, char Num);


short ROM_ReadWord(int address);
void ROM_Send(int Address, char *Data, char Num);
void ROM_Read(int Address, void *Data, char Num);
char i2c_in_byte(void);
void i2c_out_byte(char o_byte);
void i2c_nack(void);
void i2c_ack(void);
void i2c_start(void);
void i2c_stop(void);
void i2c_high_sda(void);
void i2c_low_sda(void);
void i2c_high_scl(void);
void i2c_low_scl(void);


#define TxData 0
#define SDA_PIN RC4
#define SCL_PIN RC3

#define SDA_DIR TRISC4
#define SCL_DIR TRISC3

#define I2C_DELAY 0

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#ifdef IR_RULES
#include <stddef.h>
#include "fsdinterpretertable.h"

#define IR_RULES 1

#define PRONTOFREQUENCY 1000000
#define MAXIRWORDS 80

#ifdef PIC
#include <pic18.h>
#include "mainlinepic.h"
#define IR_LED_ON       pwm_start()
#define IR_LED_OFF      pwm_stop()
#endif



void ir_initDevice(void);
void ir_LedOn(const unsigned short T);
void ir_LedOff(const unsigned short T);
void ir_Initialize(void);

long ir_CalcFrequency(const short N);
short ir_CalcOneCycle(const long frequency);







#define MAXIRCOMMAND 29







#define TITLE  0







#define MENU  1

#define PLAY  2

#define STOPDVD  3

#define PAUSE  4

#define STEP  5
```

```
#define PREVCHAPTER  6

#define NEXTCHAPTER  7


#define SEARCH  8

#define NAV_UP  9
#define NAV_DOWN  10
#define NAV_LEFT  11
#define NAV_RIGHT  12


#define REWIND  13

#define FORWARD  14
#define NUM_1  15
#define NUM_2  16
#define NUM_3  17
#define NUM_4  18
#define NUM_5  19
#define NUM_6  20
#define NUM_7  21
#define NUM_8  22
#define NUM_9  23
#define NUM_0  24
#define NUM_TEN_PLUS  25
#define POWER 26



#define MAXIRMACRO 3

#define CHAPTERSEEK 0
#define TITLESEEK 1
#define TIMESEEK 2


#define DVDDEVICE  100

#define SUBTITLE  26

#define AUDIO  27

#define ZOOM  29

#define REPEAT  32

#define SLOW 33

#define SHUFFLE  34

#define DISPLAY  35

#define PROGRAM_APEX  36
#define ANGLE  37
#define LEARN  71
#define PROGRAM_SPITFIRE  64
#define OPEN_CLOSE  13
#define SETUP_SAMPO  12
#define FourX_SPITFIRE  84
#define OneX_SPITFIRE  81

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef  __support_h_
#define __support_h_

#define PIC       1


#define FLASHAREASIZE               63000 - FLASHAREAORIGIN
#define FLASHAREAORIGIN    49152

#ifdef PIC
#define strcasecmp strcmp
#define strnocasecmp stricmp
#else
#define strcasecmp _strcmp
#define strnocasecmp _stricmp
#endif

#define DEBUG 2




#define IR_UNIV_CHIP 1


#define TEMPBUFFER_SIZE 64

#ifdef DEBUG
#include <stdio.h>
#endif
#include <stdlib.h>

#define FatalError( Str ) debug(Str); asm(" reset")
#define Error( Str )       debug(Str)
#define ErrorMsgC(Str)  debug((Str))
#define FatalMsgC(Str)  FatalError((Str))
#define FALSE 0
#define TRUE 1
#define NODE_ERROR   0x4004


typedef long                                DWORD;
typedef unsigned long               UDWORD;
typedef char                BOOL;
typedef unsigned char       BYTE;
typedef short                               WORD;
typedef unsigned short              UWORD;
typedef int                 INT;
typedef unsigned int        UINT;



#ifdef DEBUG
        #define debugPutstr(x) puts(x);
        #define debug(x) printf x; printf("\r\n");
        #if (DEBUG >= 2)
```

```
                #define debugPutstrHi(x) puts(x);
                #define debugHi(x) printf x; printf("\r\n");
        #else
                #define debugPutstrHi(x)
                #define debugHi(x)
        #endif
#else
        #define debug(x)
        #define debugHi(x)
        #define debugPutstr(x)
        #define debugPutstrHi(x)
#endif


void longToAscii (unsigned long input, char *str);


#define DIRECTORY "c:\\smarttoy\\"
#define LOGFILE   "logs\\fsdClog.txt"

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"


#include <stdarg.h>
#include <time.h>
#ifdef DEBUG
#include <stdio.h>
#endif

#ifndef PIC



void logMessage(const char *format, ...)
{
   FILE *outp;
   char temp[9];
   va_list args;

   if( (outp = fopen(DIRECTORY LOGFILE, "at")) != NULL )
   {
                va_start(args, format);
                _strdate(temp);
                fprintf(outp, "%s ", temp);
                _strtime(temp);
                fprintf(outp, "%s ", temp);

                vfprintf(outp, format, args);
                fputs( "\n", outp);
                fclose( outp );
                va_end(args);
   }
}
#endif









void longToAscii (unsigned long input, char *str)
{
  char digit, count=0, dest=0;
  char buffer[12];

  for (digit=0; digit < 12; digit++) {
    buffer[digit] = (char) ((input % 10) + '0');
    input = input / 10;
         count++;
         if (input == 0) break;
  }
  while (count-- > 0) {
```

```
            str[dest++] = buffer[count];
    }
    str[dest] = 0;
}

#ifndef PIC
void DelayMs(short ms)
{
}
#endif
```

```
Type=Exe
Reference=*\G{00020430-0000-0000-C000-000000000046}#2.0#0#..\..\..\..\WINDOWS\System32\Stdole
Reference=*\G{F5078F18-C551-11D3-89B9-0000F81FE221}#3.0#0#..\..\..\..\WINDOWS\System32\msxml3
Reference=*\G{420B2830-E718-11CF-893D-00A0C9054228}#1.0#0#..\..\..\..\WINDOWS\System32\scrrun
Class=FSDCompileScript; FSDCompileScript.cls
Module=FastSimpleDocument; FastSimpleDocument.bas
Form=Form1.frm
Startup="Form1"
ExeName32="CompileIrCodes.exe"
Command32=""
Name="CompileIrCodes"
HelpContextID="0"
CompatibleMode="0"
MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
VersionCompanyName="Systems1"
CompilationType=0
OptimizationType=0
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FlPointCheck=0
FDIVCheck=0
UnroundedFP=0
StartMode=0
Unattended=0
Retained=0
ThreadPerObject=0
MaxNumberOfThreads=1

[MS Transaction Server]
AutoRefresh=1
```

```
'
 ' PushPlay -- An Xml Document emulator\interpreter for microprocessors
 ' Copyright (C) 2002, Arthur Gravina. Confidential.
 ' Arthur Gravina <art@agravina.com>
 '
 '

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1
  Persistable = 0
  DataBindingBehavior = 0
  DataSourceBehavior  = 0
  MTSTransactionMode  = 0
END
Attribute VB_Name = "FSDCompileScript"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Public Event info(sMsg As String)

Private oXml As DOMDocument30
Private indent As Integer
Private colText As New Dictionary

Private Sub saveNodeDynamic(nodeId As Integer, node As node_def)
    If nodeId < 0 Then
        MsgBox "saveNodeDynamic: trying to save a "
        dynamicNodes(Abs(nodeId) - 2) = node
    Else
        nodes(nodeId) = node
    End If
End Sub
Private Sub setLocations()
    Dim offset As Integer
    Dim nodeSize As node_def
    Dim attrSize As attribute_def
    Dim root As Integer
    Dim temp As String
    Dim attr As Integer

    On Error GoTo errrtn
    header.nodeOffset = Len(header)
    offset = header.nodeOffset
    header.numNodes = numNodes
    offset = offset + numNodes * Len(nodeSize)
    header.attributeOffset = offset
    header.numAttributes = numAttributes
    offset = offset + numAttributes * Len(attrSize)
    header.textAreaOffset = offset
    header.lenTextArea = nextTextLoc

    root = fsd_getRootNode()
    attr = fsd_getAttributeByName(root, "scriptType")
    If (attr <> -1) Then
        header.scriptType = CInt(fsd_getAttributeValue(attr))
    Else
        header.scriptType = 0
    End If
    attr = fsd_getAttributeByName(root, "scriptId")
    If (attr <> -1) Then
        header.scriptId = CInt(fsd_getAttributeValue(attr))
    Else
```

```
        header.scriptId = 0
    End If
    Exit Sub
errrtn:
    MsgBox "setLocations Error: " & Error
End Sub

Sub fsd_writeFile(filename As String)
    Dim i As Integer
    On Error Resume Next

    On Error Resume Next
    RaiseEvent info("Writing.. " & filename)
    Kill filename
    On Error GoTo err
    setLocations
    Open filename For Binary As #1
    Put #1, 1, header

    Put #1, , nodes

    Put #1, , attributes

    ReDim Preserve textBuffer(nextTextLoc - 1)
    Put #1, , textBuffer
err:
    Close #1
End Sub


Function fsd_loadScript(sName As String, errors As String) As Boolean
    Dim ret As Boolean
    Dim root
    Dim buttonlist As IXMLDOMNodeList
    Dim commandlist As IXMLDOMNodeList
    Dim buttonNode As IXMLDOMNode
    Dim commandNode As IXMLDOMNode
    Dim sCmdName As String
    Dim iCmd As Integer
    Dim i As Integer, j As Integer

    Set oXml = Nothing
    Set oXml = New DOMDocument30
    oXml.async = False

    ret = oXml.Load(sName)
    If oXml.parseError.errorCode <> 0 Then
        With oXml.parseError
            errors = "document Parse Error:" & vbCrLf & _
                "Code: " & .errorCode & vbCrLf & _
                "Line: " & .Line & vbCrLf & _
                "lPos: " & .linepos & vbCrLf & _
                "Reason: " & .reason & vbCrLf & _
                "Src: " & .srcText & vbCrLf & _
                "fPos: " & .filepos
        End With
        fsd_loadScript = False
        Set oXml = Nothing
        Exit Function
    End If
    fsd_loadScript = True
End Function
Public Sub WalkTree()
    indent = 0
    treeWalk oXml
End Sub
```

```
Private Function attributeWalk(node As IXMLDOMNode)
    Dim i As Integer
    Dim ostr As String
    Dim attrib As IXMLDOMAttribute

    For Each attrib In node.attributes
        For i = 1 To indent
          ostr = ostr & " "
        Next
        ostr = ostr & "|--"
        ostr = ostr & attrib.nodeTypeString
        ostr = ostr & ":"
        ostr = ostr & attrib.name
        ostr = ostr & "--"
        ostr = ostr & attrib.nodeValue
        RaiseEvent info(ostr)
        ostr = ""
    Next
End Function

Private Function treeWalk(node As IXMLDOMNode)
    Dim nodeName As String
    Dim root As IXMLDOMNode
    Dim child As IXMLDOMNode
    Dim i As Integer
    Dim ostr As String

    indent = indent + 2

    For Each child In node.childNodes
      For i = 1 To indent
        ostr = ostr & " "
      Next
      ostr = ostr & "|--"
      ostr = ostr & (child.nodeTypeString)
      ostr = ostr & "--"
      If child.nodeType < 3 Then
        ostr = ostr & child.nodeName
        RaiseEvent info(ostr)
        ostr = ""
      End If

      If (child.nodeType = 1) Then
        If (child.attributes.length > 0) Then
          indent = indent + 2
          attributeWalk child
          indent = indent - 2
        End If
      End If
      If (child.hasChildNodes) Then
        treeWalk child
      Else
        ostr = ostr & child.Text
        RaiseEvent info(ostr)
        ostr = ""
      End If
    Next

  indent = indent - 2

End Function
Private Sub compileAttributeWalk(node As IXMLDOMNode, parentNodeId As Integer)
    Dim i As Integer
    Dim ostr As String
    Dim attrib As IXMLDOMAttribute
    Dim firstTime As Boolean
    Dim prevAttributeNode As Integer
```

```
    Dim attributeNode As Integer
    Dim localNode As node_def
    Dim localAttribute As attribute_def

    firstTime = True
    prevAttributeNode = -1

    For Each attrib In node.attributes
        attributeNode = addAttribute(attrib, parentNodeId)
        If prevAttributeNode <> -1 Then
            localAttribute = fetchAttribute(prevAttributeNode)
            localAttribute.nextAttribute = attributeNode
            saveAttribute prevAttributeNode, localAttribute
        End If

        If parentNodeId <> -1 And firstTime = True Then
            localNode = fetchNode(parentNodeId)
            localNode.firstAttribute = attributeNode
            saveNodeDynamic parentNodeId, localNode
        End If
        prevAttributeNode = attributeNode
        RaiseEvent info("  AddAttribute: " & attrib.nodeName & "=" & attrib.nodeValue)
        firstTime = False
    Next

End Sub
Private Sub compileWalk(node As IXMLDOMNode, parentNodeId As Integer)
    Dim root As IXMLDOMNode
    Dim child As IXMLDOMNode
    Dim i As Integer
    Dim ostr As String
    Dim nodeId As Integer
    Dim prevNodeId As Integer
    Dim firstTime As Boolean
    Dim localNode As node_def

    prevNodeId = -1
    firstTime = True
    For Each child In node.childNodes
      nodeId = addNode(child, parentNodeId)
      RaiseEvent info("Add Node: " & child.nodeName & "(" & nodeId & ")")
      If prevNodeId <> -1 Then
        localNode = fetchNode(prevNodeId)
        localNode.nextNode = nodeId
        saveNodeDynamic prevNodeId, localNode
      End If
      prevNodeId = nodeId

      If parentNodeId <> -1 And firstTime = True Then
        localNode = fetchNode(parentNodeId)
        localNode.firstChild = nodeId
        saveNodeDynamic parentNodeId, localNode
      End If
      firstTime = False



      If (child.nodeType = 1) Then
        If (child.attributes.length > 0) Then
          compileAttributeWalk child, nodeId
        End If
      End If

      If (child.hasChildNodes) Then
        compileWalk child, nodeId
      End If
    Next
```

```
End Sub
Sub fsd_Compile(inFileName As String)
    Dim totSize As Integer
    Dim nodeSize As node_def
    Dim attrSize As attribute_def

    fsd_Initialize
    colText.CompareMode = BinaryCompare
    colText.removeAll
    ReDim textBuffer(10000)
    compileWalk oXml, -1
    totSize = nextTextLoc


    totSize = totSize + (Len(nodeSize) * numNodes)
    totSize = totSize + (Len(attrSize) * numAttributes)

    RaiseEvent info("Text=" & nextTextLoc & ", Nodes=" & Len(nodeSize) * numNodes & _
            ", attributes=" & Len(attrSize) * numAttributes & ", Total=" & totSize)
    fsd_writeFile inFileName
End Sub

Private Function addNode(node As IXMLDOMNode, parentNodeId As Integer) As Integer
    Dim cNode As node_def
    Dim sName As String
    Dim nodeId As Integer

    nodeId = numNodes
    ReDim Preserve nodes(numNodes)
    numNodes = numNodes + 1
    cNode.typeNode = node.nodeType
    cNode.parentNode = parentNodeId
    cNode.nextNode = -1
    cNode.firstAttribute = -1
    cNode.firstChild = -1
    sName = node.nodeName
    cNode.locName = addCompiledText(sName)
    cNode.lenName = CByte(Len(sName))
    nodes(nodeId) = cNode
    addNode = nodeId
End Function
Private Function addAttribute(attrNode As IXMLDOMAttribute, parentNode As Integer) As Integer
    Dim attributeNode As attribute_def
    Dim sName As String, sValue As String
    Dim attrId As Integer

    attrId = numAttributes
    ReDim Preserve attributes(numAttributes)
    numAttributes = numAttributes + 1

    attributeNode.parentNode = parentNode
    attributeNode.nextAttribute = -1
    sName = attrNode.name
    sValue = attrNode.nodeValue
    attributeNode.locName = addCompiledText(sName)
    attributeNode.lenName = CByte(Len(sName))
    attributeNode.locValue = addCompiledText(sValue)
    attributeNode.lenValue = CByte(Len(sValue))

    attributes(attrId) = attributeNode
    addAttribute = attrId

End Function
Sub interpretWalk(node As Integer)
    Dim i As Integer
    Dim childCount As Integer
```

```
    Dim nodeId As Integer

    childCount = fsd_getChildCount(node)
    For i = 0 To childCount - 1
      nodeId = fsd_getNthNode(node, i)
      RaiseEvent info("Add Node: " & fsd_getNodeName(nodeId) & "(" & nodeId & ")")

      If (fsd_hasAttributes(nodeId)) Then
          interpretAttributeWalk nodeId
      End If

      If (fsd_hasChildNodes(nodeId) = True) Then
        interpretWalk nodeId
      End If
    Next

End Sub
Sub interpretAttributeWalk(node As Integer)
    Dim i As Integer
    Dim attributeCount As Integer
    Dim nodeId As Integer

    attributeCount = fsd_getAttributeCount(node)

    For i = 0 To attributeCount - 1
      nodeId = fsd_getNthAttribute(node, i)
      RaiseEvent info("  AddAttribute: " & fsd_getAttributeName(nodeId) & "=" & fsd_getAttribu
    Next

End Sub


Private Function findText(sText As String) As Integer
    On Error GoTo notfound
    If (colText.Exists(sText)) Then
        findText = colText.Item(sText)
    Else
        findText = -1
    End If
    Exit Function
notfound:
    findText = -1

End Function


Private Function addText(sText As String) As Integer
    Dim slen As Integer
    Dim loc As Integer
    Dim bt As Byte
    Dim ba() As Byte
    Dim i As Integer

    On Error GoTo errrtn

    slen = Len(sText)
    If slen = 0 Then
        addText = -1
        Exit Function
    End If
    loc = nextTextLoc
    If (loc + slen + 2) >= UBound(textBuffer) Then

        ReDim Preserve textBuffer(UBound(textBuffer) + 1024)
    End If
    ba = StringToSingleBytes(sText)
```

```
    For i = 0 To slen - 1
        textBuffer(nextTextLoc) = ba(i)
        nextTextLoc = nextTextLoc + 1
    Next i
    textBuffer(nextTextLoc) = 0
    nextTextLoc = nextTextLoc + 1
    addText = loc
    Exit Function
errrtn:
    MsgBox "Error: " & err

End Function


Private Function addCompiledText(sText As String) As Integer
    Dim slen As Integer
    Dim loc As Integer
    Dim bt As Byte
    Dim ba() As Byte
    Dim i As Integer

    On Error GoTo errrtn
    slen = Len(sText)
    If slen = 0 Then
        addCompiledText = -1
        Exit Function
    End If



    loc = findText(sText)
    If loc = -1 Then
        loc = addText(sText)
        colText.Add sText, loc
    End If
errrtn:
    addCompiledText = loc
End Function
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#include "rstack.h"

#ifdef PIC
near
#endif
static int sp=0;
static RElementType val[RMAXDIM];


void RPush(const RElementType f)
{
        if (sp<RMAXDIM) {
                val[sp++]=f;
        }
        else {
                debugPutstrHi("RSTack Oflow");
        }
}


RElementType RPop(void)
{
        if (sp>0)
                return val[--sp];
        else {
                return ISTKERROR;
        }
}


RElementType RPeek(const int Item)
{
        if (Item >= 0 && Item < sp)
                return val[sp - Item - 1];
        else {
                return ISTKERROR;
        }
}


int RCount()
{
        return sp;


}


void EmptyRStack(void)
{
        sp = 0;
}
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#include "squeue.h"
#include "fsdtablelarge.h"
#include "fsdinterpretertable.h"
#include "eprom.h"
#include "beep.h"
#include "sendircommon.h"
#include "sendirrules.h"

#ifdef PIC
        #include "i2c_ccs.h"
        #include "tablereadwrite.h"
        #include "mainlinepic.h"
        #include "delay.h"
#endif

extern const unsigned char *flashMemory;



void testFsd(void)
{

        fsdint_RunInterpreter();









}
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#include "eprom.h"

#ifdef PIC

#include "delay.h"
#include <pic18.h>
static void epromDelay(void)
{
        DelayMs(10);
}

short epromReadWord(short address)
{
        short data;

        EEPROM_READ(address);
        data = EEDATA << 8;
        address++;
        EEPROM_READ(address);
        data = data | EEDATA;
        return data;
}
void epromWriteWord(short address, short data)
{
        EEPROM_WRITE(address, data >> 8);
        epromDelay();
        address++;
        EEPROM_WRITE(address, data & 0xFF);
        epromDelay();

}
#else

#include <io.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>

#define EPROM_FILE "c:\\smarttoy\\eprom.dat"


short epromReadWord(short address)
{
        int fh;
        int ret;
        short data;

    fh =  _open(EPROM_FILE, _O_RDONLY | _O_BINARY | _O_RANDOM);
    if (fh == -1) {
        return -1;
    }


    ret = _lseek(fh, (long)address, SEEK_SET);
        if (ret != address) {
                return -1;
        }
```

```
    ret = _read( fh, &data, sizeof( data ));
         if (ret == sizeof(data) ) {
                  return data;
         }
         else {
                  return -1;
         }
}


void epromWriteWord(short address, short data)
{
         int fh;
         int ret;

    fh =  _open(EPROM_FILE, _O_RDWR | _O_BINARY | _O_CREAT | _O_RANDOM, _S_IWRITE  );
    if (fh == -1) {
        return ;
    }


    ret = _lseek(fh, (long)address, SEEK_SET);
         if (ret != address) {
                  return;
         }
    ret = _write( fh, &data, sizeof( data ));
         _close(fh);
}

#endif


short epromValid(void) {
         short marker;

         marker = epromReadWord(0);
         if (marker == NODE_ERROR) {
                  return TRUE ;
         }
         else {
                  return FALSE;
         }
}




void epromInitializeScript(short scriptNumber)
{
         struct eprom_script_def script;

         if (scriptNumber == 0) {
                  epromInitializeControl();
         }
         script.type = 0;
         script.id = 0;
         script.location = 0;
         script.len = 0;
         epromWriteScriptNumber(scriptNumber, &script);
}


void epromInitializeControl(void)
{
         epromWriteWord(EPROM_MARKER, NODE_ERROR);
         epromWriteWord(EPROM_IR_SCRIPTID, -1);
```

```
}

void epromInitialize(short bInit)
{
        short i;

        if (!epromValid() || bInit == TRUE) {
                if (bInit == FALSE) {
                        debugPutstr("epromInvalid epromInit");
                }
                for (i=0; i < EPROM_NUM_SCRIPTS; i++) {
                        epromInitializeScript(i);
                }
        }
}


void epromGetScriptNumber(short scriptNumber, struct eprom_script_def *script)
{
        short address;

        if (!epromValid()) {
                debugPutstr("epromInvalid getScriptNumber");
        }

        if (!epromValid() || scriptNumber >= EPROM_NUM_SCRIPTS || scriptNumber < 0) {
                script->type = -1;
                return;
        }

        address = (scriptNumber * sizeof(struct eprom_script_def)) + sizeof(struct eprom_cont
        script->type = epromReadWord((short)(EPROM_SCRIPT_TYPE + address));
        script->id = epromReadWord((short)(EPROM_SCRIPT_ID + address));
        script->location = epromReadWord((short)(EPROM_SCRIPT_LOCATION + address));
        script->len = epromReadWord((short)(EPROM_SCRIPT_LEN + address));
}

short epromGetScript(short scriptType, short scriptId, struct eprom_script_def *script)
{
        short i;

        for (i = 0; i < EPROM_NUM_SCRIPTS; i++) {
                epromGetScriptNumber(i, script);
                if (script->type == scriptType) {
                        if (script->id == -1 || script->id == script->id) {
                                return i;
                        }
                }
        }
        script->type = -1;
        return -1;
}


void epromWriteScriptNumber(short scriptNumber, struct eprom_script_def *script)
{
        short address;

        if (!epromValid() || scriptNumber >= EPROM_NUM_SCRIPTS || scriptNumber < 0) {
                debugPutstr("invalid epromWrite");
                return;
        }

        address = (scriptNumber * sizeof(struct eprom_script_def)) + sizeof(struct eprom_cont
        epromWriteWord((short)(EPROM_SCRIPT_TYPE + address), script->type);
        epromWriteWord((short)(EPROM_SCRIPT_ID + address), script->id);
        epromWriteWord((short)(EPROM_SCRIPT_LOCATION + address), script->location);
```

```
        epromWriteWord((short)(EPROM_SCRIPT_LEN + address), script->len);
}
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#ifdef IR_RULES
#include "fsdinterpretertable.h"

#define irdataOffset offsetFlashMemory

#define RC5_CODE           0x0001
#define RC6_CODE           0x0002
#define RCMM                       0x0004
#define SPACE_ENC          0x0008
#define REVERSE                    0x0010
#define NO_HEAD_REP                0x0020
#define NO_FOOT_REP                0x0040
#define CONST_LENGTH    0x0080
#define RAW_CODES       0x0100
#define REPEAT_HEADER   0x0200
#define SHIFT_ENC          RC5_CODE
#define SPECIAL_TRANSMITTER 0x0400

#define PULSE_BIT 0x1000000

struct flaglist {
        const char *name;
        int flag;
};

#define IR_CODE_LENGTH 2



struct ir_code_tag
{
        unsigned long data[IR_CODE_LENGTH];
        unsigned char bits[IR_CODE_LENGTH];
};

typedef struct ir_code_tag ir_code;



struct mytimeval {
        long tv_sec;
        long tv_usec;
};

struct ir_ncode {
        char *name;
        ir_code code;
        int length;
        unsigned long *signals;
};


struct ir_remote
{
        char *name;
```

```
        struct ir_ncode *codes;
        int bits;
        unsigned int flags;

        unsigned int phead,shead;
        unsigned int pthree,sthree;
        unsigned int ptwo,stwo;
        unsigned int pone,sone;
        unsigned int pzero,szero;
        unsigned int plead;
        unsigned int ptrail;
        unsigned int pfoot,sfoot;
        unsigned int prepeat,srepeat;
        int pre_data_bits;
        ir_code pre_data;
        int post_data_bits;
        ir_code post_data;
        unsigned int pre_p,pre_s;
        unsigned int post_p, post_s;
        unsigned long gap;
        unsigned long repeat_gap;
        int toggle_bit;
        unsigned int min_repeat;
        unsigned int freq;
        unsigned int duty_cycle;


    unsigned int repeat_state;
        struct ir_ncode *last_code;
        unsigned int reps;
        struct mytimeval last_send;
        unsigned long remaining_gap;
    struct ir_remote *next;
};


unsigned long s_strtoul(char *val, char **endptr, char base);


void send_space(unsigned long length);

void send_pulse(unsigned long length);


void ir_send_data_long(unsigned long value, char bits);
void ir_code_init(ir_code *code);

void ir_initPointersFromRom(short address, short len);


void ir_strtocode(char *val, char which, char numBits, ir_code *code);


void ir_set_bit(ir_code *code, short bitnum, char data);
```

```
char ir_get_bit(ir_code *code, short bitnum);


void ir_reverse(ir_code *inCode, ir_code *outCode);

void ir_send_header(struct ir_remote *remote);

void ir_LedOn(const unsigned short T);
void ir_LedOff(const unsigned short T);

void ir_sendcode(struct ir_remote *remote, char *button_name);

void send (struct ir_ncode *data,struct ir_remote *remote, unsigned short reps);

void sendRaw(unsigned long *raw, int cnt);

void ir_initWords(unsigned char command);

void ir_addWord(char flag, unsigned long word);

void ir_sendWords(unsigned char command);

void ir_endWords(unsigned char command);


void ir_configIrCodes(void);
void ir_configTest(void);

void ir_configIrCodesRom(void);

unsigned char ir_lookupButton(const char *buttonName);

void ir_sendNumbersString(const char *sNum);

NodeId ir_findMacro(short butNumber, const char *butName);
void ir_rulesInit(void);

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"

#ifdef IR_UNIV_CHIP


#include <string.h>
#include <stdio.h>
#include "fsdtablelarge.h"
#include "fsdinterpretertable.h"
#include "sendirunivchip.h"
#include "beep.h"

#ifndef PIC
#include <conio.h>
#endif

static void sndByte(unsigned char c);
#define Device_DVD  0x6000


const struct flaglist allCommands[] = {
        {"TITLE",                       TITLE},
        {"MENU",                MENU},
        {"PLAY",                PLAY},
        {"STOP",                STOPDVD},
        {"PAUSE",                       PAUSE},
        {"STEP",                STEP},
        {"PREVCHAPTER",                 PREVCHAPTER},
        {"NEXTCHAPTER",                 NEXTCHAPTER},
        {"SEARCH",                      SEARCH},
        {"NAV_UP",                      NAV_UP},
        {"NAV_DOWN",                    NAV_DOWN},
        {"NAV_LEFT",                    NAV_LEFT},
        {"NAV_RIGHT",                   NAV_RIGHT},
        {"REWIND",                      REWIND},
        {"FORWARD",                     FORWARD},
        {"NUM_1",                       NUM_1},
        {"NUM_2",                       NUM_2},
        {"NUM_3",                       NUM_3},
        {"NUM_4",                       NUM_4},
        {"NUM_5",                       NUM_5},
        {"NUM_6",                       NUM_6},
        {"NUM_7",                       NUM_7},
        {"NUM_8",                       NUM_8},
        {"NUM_9",                       NUM_9},
        {"NUM_0",                       NUM_0},
        {"NUM_TEN_PLUS",                NUM_TEN_PLUS},



        {"POWER",                       POWER},
        {NULL, 0},
};

extern short      irScriptBuffer;
```

```
static short irmacros[MAXIRMACRO+1];

static short DeviceNumber;
short                       devTicks;


static void sndByte(unsigned char c)
{
#ifdef PIC
        putch(c);
#else
        _putch(c);
#endif
}



void ir_initDevice(void)
{
        NodeId nodeDevice;
        char buffer[4];

        fsd_switchRomBuffer(irScriptBuffer);
        nodeDevice = fsd_getRootNode();
        if (nodeDevice != NODE_ERROR) {

                fsd_getAttribute(nodeDevice, "ticks", buffer, 4);
                devTicks = (short)atoi(buffer);
        }
        else {
                devTicks = -1;
        }
        debugHi(("devTicks %d node %d", devTicks, nodeDevice));

        fsd_unswitchRomBuffer();
        return;
}



void ir_Initialize(void)
{
        struct eprom_script_def script;
        short scriptType, scriptId;

        devTicks = -1;
        scriptType = IRSCRIPT;
        if (epromValid() ) {
                scriptId = epromReadWord(EPROM_IR_SCRIPTID);
        }
        else {
                scriptId = -1;
        }

        if (scriptId != -1) {
                if (epromGetScript(scriptType, scriptId, &script) == -1) {
                        fsd_setScriptBuffer(scriptType, scriptId);
                } else {
                        fsd_setScriptBufferNoLoad(&script);
                }


                ir_initDevice();
```

```
                if (devTicks == 0) devTicks = -1;
        }

        if (devTicks == -1) {
                errorBeep();
                debugPutstrHi("No ir device");
        }
}


static unsigned char getDeviceType(char pos)
{
        short dt;

        dt = Device_DVD | DeviceNumber;
        return dt >> (8 * pos);
}

void ir_setDeviceNumber(short num)
{
        DeviceNumber = num;
}

static unsigned char checkStatus(void)
{









        return TRUE;

}


unsigned char ir_sendWords(unsigned char code)
{
        unsigned char flag;

        debug(("\nir_SendKey: %d", code));
        flag = 0;
        sndByte('U');
        sndByte('I');
        sndByte('B');
        sndByte('1');
        sndByte(getDeviceType(0));
        sndByte(getDeviceType(1));
        sndByte(code);
        sndByte(flag);

        return(checkStatus());
}



void ir_sendNumbersString(const char *sNum)
{
    char sNumber;
```

```
        while((sNumber = *sNum++) > 0) {
                sNumber -= '0';
          switch (sNumber) {
              case 0:
                                    ir_sendWords(NUM_0);
                                    break;
              case 1:
                  ir_sendWords(NUM_1);
                                    break;
              case 2:
                  ir_sendWords(NUM_2);
                                    break;
              case 3:
                  ir_sendWords(NUM_3);
                                    break;
              case 4:
                  ir_sendWords(NUM_4);
                                    break;
              case 5:
                  ir_sendWords(NUM_5);
                                    break;
              case 6:
                  ir_sendWords(NUM_6);
                                    break;
              case 7:
                  ir_sendWords(NUM_7);
                                    break;
              case 8:
                  ir_sendWords(NUM_8);
                                    break;
              case 9:
                  ir_sendWords(NUM_9);
                                    break;
          }
#ifdef PIC

#endif
    }
}


unsigned char ir_lookupButton(const char *buttonName)
{
    const struct flaglist *flaglptr;
          unsigned char command;

          command = 255;
          flaglptr=allCommands;
          while(flaglptr->name!=NULL){
                  if(strnocasecmp(flaglptr->name, buttonName)==0){
                          command= flaglptr->flag;
                          break;
                  }
                  flaglptr++;
          }
          return command;
}



NodeId ir_findMacro(short butNumber, const char *butName)
{
          NodeId butLoc;
```

```
        fsd_switchRomBuffer(irScriptBuffer);
        if (butNumber >= 0 && butNumber <= MAXIRMACRO) {
                if ( irmacros[butNumber] == NODE_ERROR) {
                        butLoc = fsdint_findButton(NODE_ROOT, "IrMacro", butName);
                        irmacros[butNumber] = butLoc;
                }
                else {
                        butLoc = irmacros[butNumber];
                }
        }
        if (butLoc != NODE_ERROR) {
                butLoc = fsdint_formBufferNode(butLoc);
        }
        fsd_unswitchRomBuffer();
        return butLoc;
}

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef __fsdinterpreter_h_
#define __fsdinterpreter_h_

#include "fsdtablelarge.h"
#include "istack.h"
#include "rstack.h"
#include "squeue.h"

#define MAX_COMMANDSIZE 16
#define NUMRETURNNODES 8


void fsdint_Initialize(void);

void fsdint_RunInterpreter(void);


void fsdint_initCommands(const char *Commands[], short (*procCall) (short, NodeId, NodeId[], s

short fsdint_lookupCommand(const char *command);

void fsdint_ButtonsOffInternal(void);

void fsdint_ButtonsOnInternal(void);

void fsdint_ButtonsOff(void);

void fsdint_ButtonsOn(void);

void fsdint_executeButton(const char *sName);

NodeId fsdint_findButton(NodeId startNode, const char *sName, const char *sId) ;

void fsd_getCommandParameter(const char *name, const NodeId commandNode, char *buffer, const s

void  fsdint_interpretButton(const NodeId buttonNode);

void  fsdint_startInterpreter();

void fsdint_fetch(const char *name, char *buffer, const short len);

void fsdint_store(const char *name, const char *value);

void fsdint_increment(const char *name, const short minValue, const short maxValue);

void fsdint_append(const char *name, const char *value);
void  fsdint_delay(long seconds, long milliseconds);
void  fsdint_hardDelay(long seconds, long milliseconds);
long GetTicks(void);

NodeId fsdint_formBufferNode(NodeId inNode);

NodeId fsdint_getBufferNode(NodeId inNode);

void fsdint_Restart(void);

void fsdint_Reset(void);
```

```
#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef __config_h_
#define __config_h_



#define target_clock                PIC_CLK
#define timer_prescale              1
#define output_direction   0
#define input_direction             1
#define pwm_pin_direction  TRISC2



#define PBthres 10



#define MICROCHIP8720 1

#ifdef LABX1
#define PIC_CLK 10000000
#define TEN_MS 65536-25000+0+2
#define BLINK_ALIVE_LED RD0
#define keypad_port    PORTB
#define keypad_tris    TRISB
#define numButtons 16
#define numRows 4
#define numCols 2
#endif

#ifdef ARTBOARD
#define PIC_CLK 10000000
#define TEN_MS 65536-25000+0+2
#define BLINK_ALIVE_LED RA2
#define keypad_port    PORTF
#define keypad_tris    TRISF
#define numButtons 16
#define numRows 4
#define numCols 4
#endif

#ifdef MICROCHIP8720
#define PIC_CLK 20000000
#define TEN_MS 65536-50000+0+2
#define BLINK_ALIVE_LED RD0
#define BEEP_LED RD1
#define keypad_port    PORTF
#define keypad_tris    TRISF
#define numButtons 16
#define numRows 4
#define numCols 4
#endif


#ifdef MICRODESIGNS

#define TEN_MS 65536-25000+0+2
```

```
#define BLINK_ALIVE_LED RA1
#define numButtons 7
#endif


#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include          <pic18.h>
#include     "config.h"
#include     "mainlinepic.h"
#include          <string.h>
#include     "support.h"
#include          "squeue.h"
#include     "delay.h"
#ifdef DEBUG
#include     "serial.h"
#endif
#include     "fsdtablelarge.h"
#include     "fsdinterpretertable.h"




near char         ALIVECNT;

near char         PBCOUNT;
near long         TICKS;

near struct {
         unsigned ISC:1;
         unsigned ISA:1;
         unsigned PDONE:1;
         unsigned OLDPB:1;
         unsigned NEWPB:1;
         unsigned dummy:3;
}PBSTATEbits[numButtons];

#ifdef IR_RULES
near unsigned short _duty_1;
#endif




const unsigned char *flashMemory = (unsigned char *)FLASHAREAORIGIN;



#ifdef LCD
 const char LCDstr[] = {0x33,0x32,0x28,0x01,0x0c,0x06,0x00,0x00};
 const char StrtStr[] = {0x80, 'P', 'u', 's', 'h', ' ', 'P', 'B', ' ', 0};
 const char BYTE_1[]  = {0x80,'B','Y','T','E','=',' ',' ',' ',0};
 const char Clear1[]  = {0x80,' ',' ',' ',' ',' ',' ',' ',' ',0};
 const char Clear2[]  = {0xC0,' ',' ',' ',' ',' ',' ',' ',' ',0};
#endif



char TEMPBUFFER[TEMPBUFFER_SIZE];
```

```
char TEMPBUFFER2[64];


#ifdef LCD
void InitLCD(void);
void DisplayC(const char *tempPtr);
void DisplayV(char *tempPtr);
void T40(void);
void ByteDisplay(void);
void DisplayLine(char linenum);
void DisplayErrorMessageV(char *str);
void DisplayErrorMessageC(const char *str);
void ClearScreen(void);
#endif

void Initial(void);
void BlinkAlive(void);
void testFsd(void);

void main (void)
{
        Initial();
        debug (("MainlinePic.c 22Sep03"));
        testFsd();
}



void interrupt InterruptHandlerHigh ()
{
        unsigned char i, buttonState;
#if defined LABX1 || ARTBOARD || MICROCHIP8720
        unsigned char col, row, key, allCol;
#endif
        if (TMR0IF)
    {
        TMR0IF = 0;
        TICKS += 10;
                if (TICKS < 0) TICKS = 0;

#ifdef LABX1




        key = 0;
        for (row = 0; row < numRows; row++) {
                keypad_port = 0;


                keypad_tris = (1 << row) ^ 0xff;
                asm ("nop");
                asm ("nop");
                allCol = (keypad_port >> 4) & 0xf;

                if (allCol != 3) {
                        key = allCol;
                        key = key ^ 0xf;
                        col = 0;
                        for (i=0; i < numCols; i++) {
                                col++;
                                if (key & 1) break;
```

```
                                key = key >> 1;
                        }
                        key = (row * numCols) + col;
                        break;
                }
        }
#endif

#if defined ARTBOARD || MICROCHIP8720
        key = 0;
        for (row = 0; row < numRows; row++) {

                keypad_port = (1 << row);
                allCol = (keypad_port >> 4);
                if (allCol != 0) {
                        key = allCol;
                        col = 0;
                        for (i=0; i < numCols; i++) {
                                col++;
                                if (key & 1) {
                                        break;
                                }
                                key = key >> 1;
                        }
                        key = (row * numCols) + col;
                        break;
                }
        }

#endif

        for (i=0; i < numButtons; i++) {

#if defined LABX1 || ARTBOARD || MICROCHIP8720
                if ((key - 1) == i) {
                    buttonState = 0;
                }
                else {
                    buttonState = 1;
                }
#endif

#ifdef MICRODESIGNS


                if (i == 0)
                        buttonState = RD2;
                else if (i == 1)
                        buttonState = RC5;
                else if (i == 2)
                        buttonState = RB2;
                else if (i == 3)
                        buttonState = RB3;
                else if (i == 4)
                        buttonState = RB4;
                else if (i == 5)
                        buttonState = RB5;
                else if (i == 6)
                        buttonState = RD3;
#endif

                PBSTATEbits[i].NEWPB = buttonState;

                if (PBSTATEbits[i].OLDPB) {
                        if (!PBSTATEbits[i].NEWPB)
                                PBCOUNT = PBthres;
                }
```

```
                if (!PBSTATEbits[i].NEWPB) {
                        if (!PBCOUNT)
                                if (!PBSTATEbits[i].PDONE){
                                        PBSTATEbits[i].ISC = 1;
                                        PBSTATEbits[i].PDONE = 1;


                                }
                }
                else
                        PBSTATEbits[i].PDONE = 0;

                if (!PBSTATEbits[i].OLDPB)
                        if (PBSTATEbits[i].NEWPB) {
                                if (PBCOUNT)
                                        PBSTATEbits[i].ISA = 1;
                                PBSTATEbits[i].PDONE = 0;
                                PBCOUNT = 0;
                        }
                if (PBCOUNT)
                        PBCOUNT--;

                if (PBSTATEbits[i].NEWPB)
                        PBSTATEbits[i].OLDPB = 1;
                else
                        PBSTATEbits[i].OLDPB = 0;
        }




                BLINK_ALIVE_LED = 0;
                if (!(--ALIVECNT))              {
                        ALIVECNT = 250;
                        BLINK_ALIVE_LED = 1;
                }


        WRITETIMER0(TEN_MS);
        }

}



void Initial(void)
{
        char i;

        di();

        PIE1=0;

#ifdef ARTBOARD

        CMCON = 0x7;
        ADCON1    = 0b00001111;
        keypad_port = 0;
        keypad_tris = 0xF0;
        TRISA     = 0b11100000      ;
        TRISC     = 0b10100000      ;
        TRISE     = 0b00000000      ;
#endif
```

```
#ifdef MICROCHIP8720

        CMCON = 0x7;
        ADCON1   = 0b00001111;
        keypad_port = 0;
        keypad_tris = 0xF0;

        PORTC = 0;
        TRISC0 = 0;

        PORTD = 0;
        TRISD = 0;
#endif

#ifdef MICRODESIGNS
        ADCON1   = 0b10001110      ;
        PORTC    = 0;
        TRISA    = 0b11100001      ;
        TRISB    = 0b11111100      ;
        TRISC    = 0b10100000      ;
        TRISD    = 0b00001111      ;
        TRISE    = 0b00000000      ;
        PORTA    = 0b00010000    ;









#endif

#ifdef LABX1
        TRISD    = 0      ;

#endif


        T0CON  = 0;
        TMR0IF = 0;
        TMR0IE = 1;
        TMR0IP = 1;
        PSA = 1;
        TMR0ON = 1;

        ALIVECNT = 250;
        TICKS = 0;


#ifdef IR_RULES
        pwm_osc_init(40000, 50);
#endif


        for (i=0; i < numButtons; i++) {
                PBSTATEbits[i].ISC = 0;
                PBSTATEbits[i].ISA = 0;
                PBSTATEbits[i].PDONE = 0;
                PBSTATEbits[i].OLDPB = 1;
                PBSTATEbits[i].NEWPB = 0;
        }


#ifdef LABX1
        RBPU = 0;
#endif
```

```
#if defined MICRODESIGNS || ARTBOARD || MICROCHIP8720
        RBPU = 1;
#endif


#ifdef LCD
        InitLCD();
        DisplayC(StrtStr);
#endif

#ifdef DEBUG
        init_comms();
#endif

        IPEN = 1;

        ei();
}



void BlinkAlive()
{
        RA4 = 1;
        if (!(--ALIVECNT))          {
                ALIVECNT = 250;
                RA4 = 0;
        }
}

#ifdef IR_RULES
static void setDuty(unsigned char X)
{
        CCPR1L = (X >> 2);
        CCP1CON = (CCP1CON & 0xCF) | ((X & 3) << 4);
}



void pwm_osc_init(unsigned long pwm_osc_frequency, unsigned short pwm_osc_dutycycle)
{
        unsigned short _pr2_1;
        double x;

        x = (double)PIC_CLK / (4 * timer_prescale * (double)pwm_osc_frequency);
        _pr2_1 = (short)(x + .5) - 1;
        _duty_1 = (((_pr2_1+1) * 4) * pwm_osc_dutycycle) / 100;

    PR2     = _pr2_1 - 1;

        CCP1CON =  0x0C;

    setDuty(0);



        if (timer_prescale == 1)
                T2CON    = ( T2CON & 0xF8) | 0;
        else if (timer_prescale == 4)
                T2CON    = ( T2CON & 0xF8) | 1;
        else if (timer_prescale == 16)
                T2CON    = ( T2CON & 0xF8) | 2;
```

```
        pwm_pin_direction = output_direction;

        TMR2ON = 1;
}


void pwm_stop(void)
{
         setDuty(0);



}

void pwm_start(void)
{

        setDuty(_duty_1);

}
#endif

#ifdef LCD



void InitLCD()
{
        char currentChar;
        const char *tempPtr;

        DelayMs(100);
        RE0 = 0;
        tempPtr = LCDstr;
        while (*tempPtr) {
                currentChar = *tempPtr;
                RE1 = 1;
                PORTD = currentChar;
                RE1 = 0;
                DelayMs(10);
                currentChar <<= 4;
                RE1 = 1;
                PORTD = currentChar;
                RE1 = 0;
                DelayMs(10);
                tempPtr++;
        }
}



void T40(void)
{
        unsigned char cCount = 7;
        while (cCount)
                cCount--;
}
```

```
void DisplayC(const char *tempPtr)
{
        char currentChar;

        RE0 = 0;
        while (*tempPtr) {
                currentChar = *tempPtr;
                RE1 = 1;
                PORTD = currentChar;
                RE1 = 0;
                currentChar <<= 4;
                RE1 = 1;
                PORTD = currentChar;
                RE1 = 0;
                T40();
                RE0 = 1;
                tempPtr++;
        }
}




void DisplayV(char *tempPtr)
{
        char currentChar;

        RE0 = 0;
        while (*tempPtr) {
                currentChar = *tempPtr;
                RE1 = 1;
                PORTD = currentChar;
                RE1 = 0;
                currentChar <<= 4;
                RE1 = 1;
                PORTD = currentChar;
                RE1 = 0;
                T40();
                RE0 = 1;
                tempPtr++;
        }
}

#endif


void checkButtons(void)
{
        char i;


        di();
        for (i=0; i < numButtons; i++) {
                if (PBSTATEbits[i].ISC == 1) {
                        strcpy(TEMPBUFFER, "Button");
```

```
                        longToAscii(i, &TEMPBUFFER[6]);

                        fsdint_executeButton(TEMPBUFFER);
                        PBSTATEbits[i].ISC = 0;
                }
        }

        ei();
}

#ifdef LCD
char    COUNT;
char    TEMP;
char    TEMPBYTE;




void ByteDisplay(void)
{
        DisplayC(BYTE_1);

        COUNT = 8;
        while (COUNT) {
                TEMP = (TEMPBYTE & 0b00000001);
                TEMP |= 0x30;
                TEMPBUFFER[COUNT] = TEMP;
                TEMPBYTE = TEMPBYTE >> 1;
                COUNT--;
        }
        TEMPBUFFER[0] = 0xc0;
        TEMPBUFFER[9] = 0;
        DisplayV(TEMPBUFFER);
}



void delay_ms(long t)
{
        long start = TICKS;
        while(1) {
                if (TICKS < start) break;
                if ((TICKS - start) > t) break;
        }
}




void ClearScreen(void)
{
        DisplayC(Clear1);
        DisplayC(Clear2);
}

void DisplayLine(char linenum) {
        if (linenum == 1) {
                TEMPBUFFER[0] = (char)0x80;
        } else {
                TEMPBUFFER[0] = (char)0xc0;
        }
        DisplayV(TEMPBUFFER);
}

void DisplayErrorMessageV(char *str)
```

```
{
        char ch;
        char *p;
        short len;
        char linenum = 1;

        RA1 = 1;
        ClearScreen();
        while (linenum < 3) {
                p = &TEMPBUFFER[1];
                len = 1;
                while (1) {
                        ch = *str++;
                        if (len == 9 || ch == 0) break;
                        *p++ = ch;
                        len++;
                }
                *p = 0;
                if (linenum == 1) {
                        DisplayLine(linenum);
                        if (ch != 0) str--;
                        linenum++;
                }
                else {
                        DisplayLine(linenum);
                        linenum++;
                }
                if (ch == 0) break;
        }
        DelayS(5);
        RA1 = 0;
}


void DisplayErrorMessageC(const char *str)
{
        char temp[TEMPBUFFER_SIZE];

        ClearScreen();
        if (strlen(str) > TEMPBUFFER_SIZE - 1) {
                strncpy(temp, str, TEMPBUFFER_SIZE -1);
                temp[TEMPBUFFER_SIZE - 1] = 0;
        } else {
                strcpy(temp, str);
        }
        DisplayErrorMessageV(temp);
}
#endif
```

The following is the scripting API currently implmented.

The PushPlay basic command set
++++++++++++++++++++++++++++++++
PushPlay
Defines a new script. Must be first element in a script.

Parameters:
" scriptType. 1 = Main Script, 2 = Infrared Driver Script, 3=Compiled Infrared data
" scriptId. A unique id for this scriptType.
Example:
  <PushPlay scriptType="1" scriptId="00001">
  </PushPlay

++++++++++++++++++++++++++++++++
Button
Defines the commands that will be executed when this button is pressed.

Parameters:
" id. Button0, Button1. Button0 is the first button, Button1 is the second and so on.
A unique id is "Startup". This is executed when the script is first started.
" name. A descriptive name.
Example:
  <Button id="Startup" name="Startup">
  <Button id="Button15" name="Restart"

++++++++++++++++++++++++++++++++
Trick
Define a macro. This is a collection of commands that will be executed multiple times. You c

Parameters:
" id. The name that will be used by TrickPlay to call this macro.

Example:
  <Trick id="monkeyGraphic">

++++++++++++++++++++++++++++++++
TrickPlay
Call a Trick. Pass it any number of parameters. The commands within the macro will reference

Parameters:
" id. The name of the macro as defined by Trick.

Example:
  <TrickPlay id="monkeyGraphic">

++++++++++++++++++++++++++++++++
If
A conditional command. Will execute the block of commands if the condition is true.

Parameters:
" id. The name of a variable. May be preceded by an '@' for indirect addressing
" oper. The operation to be tested. Operators are: eq, neq, gt, lt.
" value. The value to compare to the variable.
"

Example:
  <If id="ElephantCounter" oper="eq" value="1">

++++++++++++++++++++++++++++++++
Set
Set a variable to a value.

Parameters:
" id. The name of a variable.

```
"         value. The value to compare to the variable.

 Example:
        <Set id="playstate" value="0"/>

++++++++++++++++++++++++++++++++
Increment
Will increment a variable with a range. When the maximum limit is reached will restart a minir

Parameters:
"        id.  The name of a variable.
"         min. The starting value when max is reached
"        max. The maximum value variable will be incremented to.

 Example:
        <Increment id="MonkeyCounter" min="0" max="2"/>

++++++++++++++++++++++++++++++++
Append
Append a string value to a variable

Parameters:
"        id.  The name of a variable.
"        value. The string to append.

 Example:
        <Append id="scriptId" value="1" />

++++++++++++++++++++++++++++++++
ButtonsOn
Allow a new button press to interrupt the command currently being processed.

Parameters:       none

Example:
        <ButtonsOn/>

++++++++++++++++++++++++++++++++
ButtonsOff
Don't allow a new button press to interrupt the command currently being processed.

Parameters:       none

Example:
        <ButtonsOff/>

++++++++++++++++++++++++++++++++
Sleep
Delay for a time period.

Parameters:
"        milliseconds.  The number of milliseconds to delay.
"        seconds. The number of seconds to delay.

 Example:
        <Sleep seconds="3"/>

The following commands are specific to DVD devices.


++++++++++++++++++++++++++++++++
Menu
Stops title playback and displays the top (or root) menu for the current title.
Parameters:       none
Example:          <Menu/>

++++++++++++++++++++++++++++++++
```

```
Title
Stops title playback and displays the title menu.
Parameters:       none
Example:          <Title/>

++++++++++++++++++++++++++++++++++
Resume
Returns to playback mode from menu mode at the same title position as when the menu was invok
Parameters:       none
Example:          <Resume/>

++++++++++++++++++++++++++++++++++
Back
Returns the display from a submenu to its parent menu.
Parameters:       none
Example:          <Back/>


++++++++++++++++++++++++++++++++++
Play
Causes the DVD to start playing, or resumes play of a paused item.
Parameters:       none
Example:          <Play />

++++++++++++++++++++++++++++++++++
Stop
Stops the playing of the DVD.
Parameters:       none
Example:          <Stop />

++++++++++++++++++++++++++++++++++
Pause
Pauses the playing of the chapter.
Parameters:       none
Example:          <Pause />

++++++++++++++++++++++++++++++++++
NextChapter
Seeks and plays the next chapter. Will loop.
Parameters:       none
Example:          <NextChapter />

++++++++++++++++++++++++++++++++++
PrevChapter
Seeks and plays the previous chapter. Will loop.
Parameters:       none
Example:          <PrevChapter />


++++++++++++++++++++++++++++++++++
TitleSeek
Seeks and plays the first chapter in the title. Title number is 1 to 99.
Parameter:
"         Title. The title number to seek to
Example:
          <TitleSeek title="3" />

++++++++++++++++++++++++++++++++++
ChapterSeek
Seeks and plays the chapter in the current title. Chapter number is 1 to 999.
Parameter:
"         chapter. The chapter number to seek to
Example:
          <ChapterSeek chapter="3" />

++++++++++++++++++++++++++++++++++
TimeSeek
```

```
Seeks to a specific time on the DVD. Specify hour,minute,second.
Parameter:
"       time. The hour, minute and second to seek to.
Example:
        <TimeSeek time="000757"/>

++++++++++++++++++++++++++++++++
FastForward
start fast forwarding
Parameters:       none
Example:          <FastForward />

++++++++++++++++++++++++++++++++
FastReverse
start fast reversing
Parameters:       none
Example:          <FastReverse />

++++++++++++++++++++++++++++++++
PushButton
Simulate a button press on a remote control device
Parameters:
"       id. The name of the button. Is device dependent.

Example:
        <PushButton id="SEARCH" />

++++++++++++++++++++++++++++++++
PushNumbers
Simulate pressing the number buttons.
Parameters:
"       value. The number string to send.

Example:
        <PushNumbers value="24" />

++++++++++++++++++++++++++++++++

The following is a complete script example.

<!-- World Animals No Interupting -->
<PushPlay scriptType="1" scriptId="00001">
        <!-- if 'playstate' is 0, then resume and set playstate to 1 -->
        <Trick id="checkPlaystate" >
                <!-- are we playing -->
                <If id="playstate" oper="eq" value="0">
                        <Resume/>
                        <Set id="playstate" value="1"/>
                </If>
        </Trick>

        <!-- All of the above is common to all Animals -->

        <!--              MONKEY                    -->
        <!--  Monkey Graphic-->
        <Trick id="monkeyGraphic">
                <ChapterSeek chapter="7" />
                <TimeSeek time="000716"/>
                <Sleep seconds="3"/>
        </Trick>

        <!-- Monkey live -->
        <Trick id="monkeyLive">
                <ChapterSeek chapter="7" />
                <TimeSeek time="000757"/>
                <Sleep seconds="50"/>
        </Trick>
```

```
<!-- M5 Monkey Puppet Sequence -->
<Trick id="monkeyPuppet">
    <ChapterSeek chapter="7" />
        <TimeSeek time="000740"/>
        <Sleep seconds="16"/>
</Trick>

<!--                        FISH                        -->
<!-- Fish Live -->

<!-- Fish -->
<Trick id="fishGraphic">
        <TimeSeek time="001055"/>
        <Sleep seconds="4"/>
</Trick>

<Trick id="fishLive">
        <TimeSeek time="001244"/>
        <Sleep seconds="41"/>
</Trick>

<Trick id="fishPuppet">
        <TimeSeek time="001356"/>
        <Sleep seconds="26"/>
</Trick>

<!--           TROPICAL BIRD              -->
<!-- Tropical Bird Graphic -->
<Trick id="tropicalBirdGraphic">
        <TimeSeek time="000326"/>
        <Sleep seconds="3"/>
</Trick>

<!-- TB3 Tropical Bird LIve -->
<Trick id="tropicalBirdLive">
        <TimeSeek time="000552"/>
        <Sleep seconds="62"/>
</Trick>

<!-- TropicalBird Puppet -->
<Trick id="tropicalBirdPuppet">
        <TimeSeek time="000655"/>
        <Sleep seconds="20"/>
</Trick>

<!--              SEA TURTLE                -->

<!--  SeaTurtle Graphic -->
<Trick id="turtleGraphic">
        <TimeSeek time="001107"/>
        <Sleep seconds="3"/>
</Trick>

<!-- Sea Turtle Live -->
<Trick id="turtleLive">
        <TimeSeek time="001141"/>
        <Sleep seconds="30"/>
</Trick>

<!-- Sea Turtle Puppet -->
<Trick id="turtlePuppet">
        <TimeSeek time="001112"/>
        <Sleep seconds="18"/>
</Trick>
```

```
        <!--              ELEPHANT                  -->
        <!--  Elephant Graphic -->
<Trick id="elephantGraphic">
                <TimeSeek time="001512"/>
                <Sleep seconds="3"/>
        </Trick>

        <Trick id="elephantLive">
                <TimeSeek time="001600"/>
                <Sleep seconds="61"/>
        </Trick>

        <Trick id="elephantPuppet">
                <TimeSeek time="001518"/>
                <Sleep seconds="17"/>
        </Trick>

        <Button id="Startup" name="Startup">
                <Set id="ElephantCounter" value="0"/>
                <Set id="SeaTurtleCounter" value="0"/>
                <Set id="TropicalBirdCounter" value="0"/>
                <Set id="MonkeyCounter" value="0"/>
                <Set id="FishCounter" value="0"/>
                <TitleSeek title="2" />
                <Sleep seconds="1" />
                <ChapterSeek chapter="4" />
        </Button>

        <!-- Restart Game. This will clear everything, and startover -->
        <Button id="Button15" name="Restart">
                <Restart />
        </Button>

        <!-- Reset Game. 1st time clear Gamescript and startover. -->
        <Button id="Button14" name="Reset">
                <Reset />
        </Button>

        <!-- Get Ir Script. -->
        <Button id="Button13" name="GetIrScript">
                <GetIrScript />
        </Button>

        <!-- Monkey Button -->
        <Button id="Button0" name="Monkey">
        <!-- <ButtonsOff/> -->

        <!-- incriment the counter pre-trickplay -->
        <Increment id="MonkeyCounter" min="0" max="2"/>

                <!-- first time  -->
                <If id="MonkeyCounter" oper="eq" value="0">
                        <TrickPlay id="monkeyGraphic" />

                        <Pause/>
                        <Set id="playstate" value="0"/>

                </If>

                <!-- second time  -->
                <If id="MonkeyCounter" oper="eq" value="1">

                        <TrickPlay id="monkeyLive" />

                        <Pause/>
                        <Set id="playstate" value="0"/>
```

```
        </If>

        <!-- third time  -->
        <If id="MonkeyCounter" oper="eq" value="2">

                <TrickPlay id="monkeyPuppet" />

                <Pause/>
                <Set id="playstate" value="0"/>
        </If>


        <!-- <ButtonsOn/> -->
</Button>

        <!-- Fish Button -->
<Button id="Button1" name="Fish">
<!-- <ButtonsOff/> -->

        <Increment id="FishCounter" min="0" max="2"/>

        <!-- first time  -->
        <If id="FishCounter" oper="eq" value="0">

                <TrickPlay id="fishGraphic" />

                <Pause/>
                <Set id="playstate" value="0"/>

        </If>

        <!-- second time  -->
        <If id="FishCounter" oper="eq" value="1">

                <TrickPlay id="fishLive" chapter="16" seconds="3" />

                <Pause/>
                <Set id="playstate" value="0"/>
        </If>

        <!-- third time  -->
        <If id="FishCounter" oper="eq" value="2">

                <TrickPlay id="fishPuppet" />

                <Pause/>
                <Set id="playstate" value="0"/>
        </If>


        <!-- <ButtonsOn/> -->
</Button>


<!-- TropicalBird Button -->
<Button id="Button2" name="Tropical Bird">
<!-- <ButtonsOff/> -->

        <Increment id="TropicalBirdCounter" min="0" max="2"/>

        <!-- first time  -->
        <If id="TropicalBirdCounter" oper="eq" value="0">

                <TrickPlay id="tropicalBirdGraphic" />

                <Pause/>
                <Set id="playstate" value="0"/>
```

```
        </If>

        <!-- second time  -->
        <If id="TropicalBirdCounter" oper="eq" value="1">

                <TrickPlay id="tropicalBirdLive" />

                <Pause/>
                <Set id="playstate" value="0"/>
        </If>

        <!-- third time  -->
        <If id="TropicalBirdCounter" oper="eq" value="2">

                <TrickPlay id="tropicalBirdPuppet" />

                <Pause/>
                <Set id="playstate" value="0"/>
        </If>


        <!-- <ButtonsOn/> -->
</Button>

<!-- SeaTurtle Button -->
<Button id="Button3" name="Sea Turtle">
<!-- <ButtonsOff/> -->

        <Increment id="SeaTurtleCounter" min="0" max="2"/>

        <!-- first time  -->
        <If id="SeaTurtleCounter" oper="eq" value="0">

                <TrickPlay id="turtleGraphic" />

                <Pause/>
                <Set id="playstate" value="0"/>

        </If>

        <!-- second time  no Guess for SeaTurtle ???? -->
        <If id="SeaTurtleCounter" oper="eq" value="1">

                <TrickPlay id="turtleLive" />

                <Pause/>
                <Set id="playstate" value="0"/>
        </If>

        <!-- third time  -->
        <If id="SeaTurtleCounter" oper="eq" value="2">

                <TrickPlay id="turtlePuppet" />

                <Pause/>
                <Set id="playstate" value="0"/>
        </If>

        <!-- <ButtonsOn/> -->
</Button>

<!-- Elephant Button -->
<Button id="Button4" name="Elephant">
<!-- <ButtonsOff/> -->

        <Increment id="ElephantCounter" min="0" max="2"/>
```

```
                <!-- first time  -->
                <If id="ElephantCounter" oper="eq" value="0">

                        <TrickPlay id="elephantGraphic" />

                        <Pause/>
                        <Set id="playstate" value="0"/>

                </If>

                <!-- second time  -->
                <If id="ElephantCounter" oper="eq" value="1">

                        <TrickPlay id="elephantLive" />

                        <Pause/>
                        <Set id="playstate" value="0"/>
                </If>

                <!-- third time  -->
                <If id="ElephantCounter" oper="eq" value="2">

                        <TrickPlay id="elephantPuppet" />

                        <Pause/>
                        <Set id="playstate" value="0"/>
                </If>

                <!-- <ButtonsOn/> -->
        </Button>

</PushPlay>
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef __rstack_h_
#define __rstack_h_

#define RMAXDIM 30
#define ISTKERROR -3333
typedef short RElementType;


void RPush(const RElementType f);

RElementType RPop(void);

RElementType RPeek(const int Item);

int RCount();

void EmptyRStack(void);

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef __eprom_h_
#define __eprom_h_

#include "support.h"
#include <stddef.h>

#define EPROM_NUM_SCRIPTS 3

struct eprom_script_def {
        WORD type;
        WORD id;
        WORD location;
        WORD len;
};

#define EPROM_SCRIPT_TYPE          offsetof(struct eprom_script_def, type)
#define EPROM_SCRIPT_ID                  offsetof(struct eprom_script_def, id)
#define EPROM_SCRIPT_LOCATION      offsetof(struct eprom_script_def, location)
#define EPROM_SCRIPT_LEN           offsetof(struct eprom_script_def, len)

struct eprom_control_def {
        WORD marker;
        WORD irScriptId;
};

#define EPROM_MARKER                     offsetof(struct eprom_control_def, marker)
#define EPROM_IR_SCRIPTID          offsetof(struct eprom_control_def, irScriptId)


short epromValid(void);
void epromInitializeScript(short scriptNumber);
void epromInitializeControl(void);
void epromInitialize(short bInit);
void epromWriteWord(short address, short data);
short epromReadWord(short address);
void epromGetScriptNumber(short scriptNumber, struct eprom_script_def *script);
short epromGetScript(short scriptType, short scriptId, struct eprom_script_def *script);
void epromWriteScriptNumber(short scriptNumber, struct eprom_script_def *script);

#endif
```

The following files contain the compiler for PushPlay: CompileIrCodes.vbp; CompileIrCodes.vb

These files are meant to be compiled under Visual Basic 6.0.
The use interface is self explanatory.

Navigate to the directory desired, where the PushPlay scripts are stored.

Click on the Compile Button.

All files will be compiled into PushPlay's proprietary format.

The file will have a ".fsd" appended to them.

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#ifdef IR_UNIV_CHIP

void ir_Initialize(void);
void ir_setDeviceNumber(short num);
unsigned char ir_sendWords(unsigned char code);
void ir_sendNumbersString(const char *sNum);
unsigned char ir_lookupButton(const char *buttonName);
NodeId ir_findMacro(short butNumber, const char *butName);


struct flaglist {
        const char *name;
        int flag;
};

#define MAXIRCOMMAND 29




#define TITLE                           35




#define MENU                            33

#define PLAY                            24

#define STOPDVD                         25

#define PAUSE                           26

#define STEP                            0

#define PREVCHAPTER                     31

#define NEXTCHAPTER                     30


#define SEARCH                          32

#define NAV_UP                          38
#define NAV_DOWN                39
#define NAV_LEFT                40
#define NAV_RIGHT               41


#define REWIND                          27
```

```
#define FORWARD                         28
#define NUM_1                           9
#define NUM_2                           10
#define NUM_3                           11
#define NUM_4                           12
#define NUM_5                           13
#define NUM_6                           14
#define NUM_7                           15
#define NUM_8                           16
#define NUM_9                           17
#define NUM_0                           18
#define NUM_TEN_PLUS            20
#define POWER                           1


#define MAXIRMACRO 3

#define CHAPTERSEEK 0
#define TITLESEEK 1
#define TIMESEEK 2

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include <pic18.h>
#include "tablereadwrite.h"

extern char TEMPBUFFER2[];
static void fsd_initiate_write(void);
static void fsd_flash_write(far unsigned char * source_addr,unsigned char length,far unsigned
static unsigned char fsd_flash_read(unsigned long addr);

static void fsd_initiate_write(void)
{
                                        WREN=1;
                                        CARRY=0;if(GIE)CARRY=1;GIE=0;
                                        DC=0;if(PEIE)DC=1;PEIE=0;
                                        EECON2=0x55;
                                        EECON2=0xAA;
                                        WR=1;
                                        asm("\tNOP");
                                        if(CARRY)GIE=1;
                                        if(DC)PEIE=1;
                                        WREN=0;
}

static void fsd_flash_write(far unsigned char * source_addr,unsigned char length,far unsigned
{

        unsigned char index;
        unsigned char offset;
#if defined(_18F242) || defined(_18F252) || defined(_18F442) || defined(_18F452)
        unsigned char saved1,saved2,saved3;
#endif

        offset=(unsigned char)dest_addr & 0x3F;
        dest_addr-=offset;

        while(length)
        {

                for(index=0;index<64;index++)
                {
                        if((index>=offset)&&(length))
                        {
                                TEMPBUFFER2[index]=*(source_addr++);
                                length--;
                        }
                        else
                                TEMPBUFFER2[index]=*(dest_addr+index);
                }


                TBLPTR=dest_addr;
                EECON1=0x94;
                fsd_initiate_write();


                for(index=0;index<64;index++)
                {
                        TABLAT=TEMPBUFFER2[index];
#if defined(_18F242) || defined(_18F252) || defined(_18F442) || defined(_18F452)
```

```
                        saved1=INTCON; INTCON=0;
                        saved2=INTCON2; INTCON2=0;
                        saved3=INTCON3; INTCON3=0;
                        TEMPBUFFER2[0]=PIE1; PIE1=0;
                        offset=PIE2; PIE2=0;
#endif
                        if(index==0)
                                asm("\tTBLWT*");
                        else
                                asm("\tTBLWT+*");

#if defined(_18F242) || defined(_18F252) || defined(_18F442) || defined(_18F452)
                        INTCON=saved1;
                        INTCON2=saved2;
                        INTCON3=saved3;
                        PIE1=TEMPBUFFER2[0];
                        PIE2=offset;
#endif
                        if((index & 7)==7)
                        {
                                fsd_initiate_write();
                        }
                }
                dest_addr+=64;
                offset=0;
        }
}

static unsigned char fsd_flash_read(unsigned long addr)
{
        TBLPTRL=((addr)&0xFF);
        TBLPTRH=(((addr)>>8)&0xFF);
        TBLPTRU=(((addr)>>8)>>8);
        asm("\tTBLRD*+");
        return TABLAT;
}


void TableWrite(unsigned char *dest, unsigned char *source, unsigned short Count)
{
        unsigned short index=0;
        unsigned char thisCount;


        while(index < Count) {

                if ((index + 64) <= Count) {
                        thisCount = 64;
                        index += 64;
                }
                else {
                        thisCount = Count - index;
                        index += Count - index ;

                }

                fsd_flash_write(source, thisCount, dest );
                source += thisCount;
                dest += thisCount;
        }
}


void TableRead(unsigned char *dest, unsigned char *source, unsigned  short Count)
```

```
{
        unsigned char data;

        while(Count > 0) {
                data = fsd_flash_read((unsigned long)source++);
                *dest++ = data;
                Count--;
        }
}
```

The following files contain the compiler for PushPlay: CompileIrCodes.vbp; CompileIrCodes

These files are meant to be compiled under Visual Basic 6.0.
The use interface is self explanatory.

Navigate to the directory desired, where the PushPlay scripts are stored.

Click on the Compile Button.

All files will be compiled into PushPlay's proprietary format.

The file will have a ".fsd" appended to them.

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#ifdef IR_UNIV_CHIP

void ir_Initialize(void);
void ir_setDeviceNumber(short num);
unsigned char ir_sendWords(unsigned char code);
void ir_sendNumbersString(const char *sNum);
unsigned char ir_lookupButton(const char *buttonName);
NodeId ir_findMacro(short butNumber, const char *butName);


struct flaglist {
        const char *name;
        int flag;
};


#define MAXIRCOMMAND 29









#define TITLE                           35




#define MENU                            33

#define PLAY                            24

#define STOPDVD                         25

#define PAUSE                           26

#define STEP                            0

#define PREVCHAPTER                     31

#define NEXTCHAPTER                     30


#define SEARCH                          32

#define NAV_UP                          38
#define NAV_DOWN                        39
```

```
#define NAV_LEFT                        40
#define NAV_RIGHT                       41


#define REWIND                          27

#define FORWARD                         28
#define NUM_1                           9
#define NUM_2                           10
#define NUM_3                           11
#define NUM_4                           12
#define NUM_5                           13
#define NUM_6                           14
#define NUM_7                           15
#define NUM_8                           16
#define NUM_9                           17
#define NUM_0                           18
#define NUM_TEN_PLUS            20
#define POWER                           1



#define MAXIRMACRO 3

#define CHAPTERSEEK 0
#define TITLESEEK 1
#define TIMESEEK 2

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include <pic18.h>
#include "tablereadwrite.h"

extern char TEMPBUFFER2[];
static void fsd_initiate_write(void);
static void fsd_flash_write(far unsigned char * source_addr,unsigned char length,far unsig
static unsigned char fsd_flash_read(unsigned long addr);

static void fsd_initiate_write(void)
{
                                WREN=1;
                                CARRY=0;if(GIE)CARRY=1;GIE=0;
                                DC=0;if(PEIE)DC=1;PEIE=0;
                                EECON2=0x55;
                                EECON2=0xAA;
                                WR=1;
                                asm("\tNOP");
                                if(CARRY)GIE=1;
                                if(DC)PEIE=1;
                                WREN=0;
}

static void fsd_flash_write(far unsigned char * source_addr,unsigned char length,far unsig
{

        unsigned char index;
        unsigned char offset;
#if defined(_18F242) || defined(_18F252) || defined(_18F442) || defined(_18F452)
        unsigned char saved1,saved2,saved3;
#endif

        offset=(unsigned char)dest_addr & 0x3F;
        dest_addr-=offset;

        while(length)
        {

                for(index=0;index<64;index++)
                {
                        if((index>=offset)&&(length))
                        {
                                TEMPBUFFER2[index]=*(source_addr++);
                                length--;
                        }
                        else
                                TEMPBUFFER2[index]=*(dest_addr+index);
                }


                TBLPTR=dest_addr;
                EECON1=0x94;
                fsd_initiate_write();
```

```
                for(index=0;index<64;index++)
                {
                        TABLAT=TEMPBUFFER2[index];
#if defined(_18F242) || defined(_18F252) || defined(_18F442) || defined(_18F452)


                        saved1=INTCON; INTCON=0;
                        saved2=INTCON2; INTCON2=0;
                        saved3=INTCON3; INTCON3=0;
                        TEMPBUFFER2[0]=PIE1; PIE1=0;
                        offset=PIE2; PIE2=0;
#endif
                        if(index==0)
                                asm("\tTBLWT*");
                        else
                                asm("\tTBLWT+*");

#if defined(_18F242) || defined(_18F252) || defined(_18F442) || defined(_18F452)
                        INTCON=saved1;
                        INTCON2=saved2;
                        INTCON3=saved3;
                        PIE1=TEMPBUFFER2[0];
                        PIE2=offset;
#endif
                        if((index & 7)==7)
                        {
                                fsd_initiate_write();
                        }
                }
                dest_addr+=64;
                offset=0;
        }
}

static unsigned char fsd_flash_read(unsigned long addr)
{
        TBLPTRL=((addr)&0xFF);
        TBLPTRH=(((addr)>>8)&0xFF);
        TBLPTRU=(((addr)>>8)>>8);
        asm("\tTBLRD*+");
        return TABLAT;
}



void TableWrite(unsigned char *dest, unsigned char *source, unsigned short Count)
{
        unsigned short index=0;
        unsigned char thisCount;


        while(index < Count) {

                if ((index + 64) <= Count) {
                        thisCount = 64;
                        index += 64;
                }
                else {
                        thisCount = Count - index;
```

```
                        index += Count - index ;

                }

                fsd_flash_write(source, thisCount, dest );
                source += thisCount;
                dest += thisCount;
        }
}


void TableRead(unsigned char *dest, unsigned char *source, unsigned  short Count)
{
                unsigned char data;

                while(Count > 0) {
                        data = fsd_flash_read((unsigned long)source++);
                        *dest++ = data;
                        Count--;
                }
}
```

```
FSDCompileScript = 72, 7, 685, 428,
FastSimpleDocument = 120, 134, 733, 555,
Form1 = 66, 87, 679, 508, Z, 21, 4, 634, 425, C
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#include "istack.h"
#ifdef PIC
near
#endif
static short sp=0;
static ElementType val[MAXDIM];


void IPush(const ElementType f)
{
        if (sp<MAXDIM) {
                val[sp++]=f;
        }
        else {
                debugPutstrHi("ISTack Oflow");
        }
}


ElementType IPop(void)
{
        if (sp>0)
                return val[--sp];
        else {
                return ISTKERROR;
        }
}


ElementType IPeek(const ElementType Item)
{
        if (Item >= 0 && Item < sp)
                return val[sp - Item - 1];
        else {
                return ISTKERROR;
        }
}


short ICount()
{
        return sp;


}


void EmptyIStack(void)
{
        sp = 0;
}
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */


#ifndef __DELAY_C
#define __DELAY_C

#include <pic18.h>

unsigned char delayus_variable;

#include "delay.h"

void DelayBigUs(unsigned int cnt)
{
        unsigned char     i;

        i = (unsigned char)(cnt>>8);
        while(i>=1)
        {
                i--;
                DelayUs(253);
                CLRWDT();
        }
        DelayUs((unsigned char)(cnt & 0xFF));
}


void DelayMs(unsigned int cnt)
{
        unsigned char     i;
        do {
                i = 4;
                do {
                        DelayUs(250);
                        CLRWDT();
                } while(--i);
        } while(--cnt);
}

void DelayS(unsigned char cnt)
{
        unsigned char i;
        do {
                i = 4;
                do {
                        DelayMs(250);
                        CLRWDT();
                } while(--i);
        } while(--cnt);
}

#endif
```

```
'
 ' PushPlay -- An Xml Document emulator\interpreter for microprocessors
 ' Copyright (C) 2002, Arthur Gravina. Confidential.
 ' Arthur Gravina <art@agravina.com>
 '
 '


Attribute VB_Name = "FastSimpleDocument"
Option Explicit

Private Declare Sub CopyMemory Lib "kernel32" Alias _
    "RtlMoveMemory" (dest As Any, source As Any, _
    ByVal numBytes As Long)

Public Const NODE_AVAILABLE = 0
Public Const NODE_DYNAMIC = &HF00
Public Const MAINSCRIPT = 1
Public Const IRSCRIPT = 2


Public header As header_def
Public nodes() As node_def
Public numNodes As Integer
Public attributes() As attribute_def
Public numAttributes As Integer

Public textBuffer() As Byte
Public nextTextLoc As Integer


Public dynamicNodes(20) As node_def
Public maxNode As Integer
Public dynamicAttributes(60) As attribute_def
Public maxAttribute As Integer
Public dynamicTextBlocks(3200) As Byte
Public maxTextBlock As Integer
Public Const TEXT_CHUNK = 32

Type header_def
    nodeOffset As Integer
    numNodes As Integer
    attributeOffset As Integer
    numAttributes As Integer
    textAreaOffset As Integer
    lenTextArea As Integer
    scriptType As Integer
    scriptId As Integer
End Type


Type node_def
    parentNode As Integer
    typeNode As Integer
    nextNode As Integer
    firstChild As Integer
    firstAttribute As Integer
    locName As Integer
    lenName As Byte
    filler As Byte
End Type


Type attribute_def
    parentNode As Integer
    nextAttribute As Integer
    locName As Integer
```

```
    locValue As Integer
    lenName As Byte
    lenValue As Byte
End Type
Function ByteArrayToString(byteArr() As Byte, StartIndex As Integer, length As Integer) As St:
    Dim res As String
    res = Space(length)
    CopyMemory ByVal res, byteArr(StartIndex), length
    ByteArrayToString = res
End Function

Function StringToSingleBytes(source As String) As Byte()
    StringToSingleBytes = StrConv(source, vbFromUnicode)
End Function
Function isArrayEmpty(arr As Variant) As Boolean
    Dim i
    isArrayEmpty = True
    On Error Resume Next
    i = UBound(arr)
    If Err.Number > 0 Then Exit Function
    isArrayEmpty = False
End Function


Function fetchNode(nodeId As Integer) As node_def
    If nodeId < 0 Then
        fetchNode = dynamicNodes(Abs(nodeId) - 2)
    Else
        fetchNode = nodes(nodeId)
    End If

End Function
Sub saveNode(nodeId As Integer, node As node_def)
    If nodeId < 0 Then
      ' dynamicNodes(Abs(nodeId) - 2) = node
    Else
        MsgBox "saveNode Error: "
        nodes(nodeId) = node
    End If
End Sub

Function fetchAttribute(attributeId As Integer) As attribute_def
    If attributeId < 0 Then
        fetchAttribute = dynamicAttributes(Abs(attributeId) - 2)
    Else
        fetchAttribute = attributes(attributeId)
    End If

End Function
Sub saveAttribute(attributeId As Integer, attr As attribute_def)

    If attributeId < 0 Then
        dynamicAttributes(Abs(attributeId) - 2) = attr
    Else
        attributes(attributeId) = attr
    End If
End Sub

Function fsd_slotAttribute() As Integer


    Dim i
    On Error GoTo errrtn
tryagain:

    For i = 0 To UBound(dynamicAttributes)
        If dynamicAttributes(i).parentNode = NODE_AVAILABLE Then
```

```
            If i > maxAttribute Then maxAttribute = i
            dynamicAttributes(i).parentNode = NODE_DYNAMIC
            dynamicAttributes(i).locName = 0
            dynamicAttributes(i).locValue = 0
            dynamicAttributes(i).nextAttribute = -1
            fsd_slotAttribute = -(i + 2)
            Exit Function
        End If
    Next i




errrtn:
    fsd_slotAttribute = 0
End Function

Function fsd_slotTextBlock() As Integer
    Dim loc As Integer

    Do While loc < UBound(dynamicTextBlocks)
        If dynamicTextBlocks(loc) = 0 And dynamicTextBlocks(loc + 1) = 0 Then

            If loc > maxTextBlock Then maxTextBlock = loc


            fsd_slotTextBlock = -(loc + 2)
            Exit Function
        End If
        loc = loc + TEXT_CHUNK
    Loop
    fsd_slotTextBlock = 0
End Function
Function fsd_slotNode() As Integer


    Dim i As Integer
    On Error GoTo errrtn
tryagain:

    For i = 0 To UBound(dynamicNodes)
        If dynamicNodes(i).typeNode = NODE_AVAILABLE Then

            If i > maxNode Then maxNode = i

            dynamicNodes(i).typeNode = NODE_DYNAMIC
            dynamicNodes(i).firstAttribute = -1
            dynamicNodes(i).firstAttribute = -1
            dynamicNodes(i).locName = 0
            dynamicNodes(i).nextNode = -1
            dynamicNodes(i).parentNode = -1
            fsd_slotNode = -(i + 2)
            Exit Function
        End If
    Next i
```

```
errrtn:
    fsd_slotNode = 0
End Function

Sub fsd_scratchNode(nodeId As Integer)
    Dim attributeNodes() As Integer
    Dim i As Integer
    Dim node As node_def


    If Not nodeId < 0 Then
        MsgBox "scratchNode Error: Trying to scratch readonly"
        Exit Sub
    End If


    attributeNodes = fsd_getAttributes(nodeId)
    If Not isArrayEmpty(attributeNodes) Then
        For i = 0 To UBound(attributeNodes)
            fsd_scratchAttribute (attributeNodes(i))
        Next i
    End If


    node = fetchNode(nodeId)
    node.typeNode = NODE_AVAILABLE
    saveNode nodeId, node

End Sub

Function fsd_scratchAttribute(nodeId As Integer)
    Dim node As Integer

    node = Abs(nodeId) - 2

    If nodeId < 0 Then
        fsd_scratchTextBlock dynamicAttributes(node).locName
        fsd_scratchTextBlock dynamicAttributes(node).locValue
        dynamicAttributes(node).parentNode = NODE_AVAILABLE
    End If

End Function
Function fsd_scratchTextBlock(loc As Integer)
    Dim newLoc As Integer
    newLoc = Abs(loc) - 2
    If loc < 0 Then
        dynamicTextBlocks(newLoc) = 0
        dynamicTextBlocks(newLoc + 1) = 0
    End If
End Function

Sub fsd_Initialize()
    numNodes = 0
    numAttributes = 0
    nextTextLoc = 0
    ReDim nodes(numNodes)
    ReDim attributes(numAttributes)
    ReDim textBuffer(nextTextLoc)
End Sub



Function fsd_addText(sText As String, Optional dynamicText As Boolean = False) As Integer
    Dim slen As Integer
    Dim loc As Integer
```

```
    Dim bt As Byte
    Dim ba() As Byte
    Dim i As Integer
    Dim nextLoc As Integer

    On Error GoTo errrtn
    slen = Len(sText)
    If slen = 0 Or slen > (TEXT_CHUNK - 2) Then
        fsd_addText = 0
        Exit Function
    End If
    loc = fsd_slotTextBlock()
    If loc = 0 Then
        MsgBox "addText Failed. no more room"
        fsd_addText = 0
        Exit Function
    End If




    nextLoc = Abs(loc) - 2
    ba = StringToSingleBytes(sText)

    For i = 0 To slen - 1
        dynamicTextBlocks(nextLoc) = ba(i)
        nextLoc = nextLoc + 1
    Next i
    dynamicTextBlocks(nextLoc) = 0
    nextLoc = nextLoc + 1
    fsd_addText = loc
    Exit Function
errrtn:
    MsgBox "Error: " & Err
    fsd_addText = 0
End Function

Function fsd_getText(locText As Integer) As String
    Dim start As Integer
    Dim slen As Integer
    Dim thisLoc As Integer

    If locText < 0 Then

        thisLoc = Abs(locText) - 2
        start = thisLoc
        Do While dynamicTextBlocks(start) <> 0
            slen = slen + 1
            start = start + 1
            If start > 1000 Then Exit Do
        Loop
        start = thisLoc
        fsd_getText = ByteArrayToString(dynamicTextBlocks, start, slen)
    Else
        start = locText
        Do While textBuffer(start) <> 0
            slen = slen + 1
            start = start + 1
            If start > 1000 Then Exit Do
        Loop
        start = locText
        fsd_getText = ByteArrayToString(textBuffer, start, slen)
    End If
End Function
```

```
Function fsd_getChildCount(nodeId As Integer) As Integer
    Dim id As Integer
    Dim cnt As Integer
    On Error GoTo errrtn
    id = fetchNode(nodeId).firstChild

    Do While id <> -1
        cnt = cnt + 1
        id = fetchNode(id).nextNode
    Loop
errrtn:
    fsd_getChildCount = cnt
End Function


Function fsd_getNthNode(nodeId As Integer, nodeNum As Integer) As Integer
    Dim id As Integer
    Dim cnt As Integer
    On Error GoTo errrtn
    id = fetchNode(nodeId).firstChild

    Do While id <> -1
        If cnt = nodeNum Then
            fsd_getNthNode = id
            Exit Function
        End If
        cnt = cnt + 1
        id = fetchNode(id).nextNode
    Loop
errrtn:
    fsd_getNthNode = id
End Function


Function fsd_hasChildNodes(nodeId As Integer) As Boolean
    On Error Resume Next
    fsd_hasChildNodes = fetchNode(nodeId).firstChild <> -1
End Function


Function fsd_getNodesByName(nodeId As Integer, sName As String) As Integer()
    Dim id As Integer
    Dim cnt As Integer
    Dim nodesFound() As Integer
    Dim cntNodesFound As Integer
    Dim sNodeName As String

    On Error GoTo errrtn
    id = fetchNode(nodeId).firstChild

    Do While id <> -1
        sNodeName = fsd_getText(fetchNode(id).locName)
        If sNodeName = sName Then
            ReDim Preserve nodesFound(cntNodesFound)
            nodesFound(cntNodesFound) = id
            cntNodesFound = cntNodesFound + 1
        End If
        id = fetchNode(id).nextNode
    Loop
errrtn:
    fsd_getNodesByName = nodesFound
End Function

Function fsd_getAttributes(parentNode As Integer) As Integer()
    Dim id As Integer
```

```
    Dim cnt As Integer
    Dim nodesFound() As Integer
    Dim cntNodesFound As Integer
    Dim sNodeName As String

    On Error GoTo errrtn
    id = fetchNode(parentNode).firstAttribute

    Do While id <> -1
        ReDim Preserve nodesFound(cntNodesFound)
        nodesFound(cntNodesFound) = id
        cntNodesFound = cntNodesFound + 1
        id = fetchAttribute(id).nextAttribute
    Loop
errrtn:
    fsd_getAttributes = nodesFound

End Function

Function fsd_getChildNodes(nodeId As Integer) As Integer()
    Dim id As Integer
    Dim cnt As Integer
    Dim nodesFound() As Integer
    Dim cntNodesFound As Integer
    Dim sNodeName As String

    On Error GoTo errrtn
    id = fetchNode(nodeId).firstChild

    Do While id <> -1
        ReDim Preserve nodesFound(cntNodesFound)
        nodesFound(cntNodesFound) = id
        cntNodesFound = cntNodesFound + 1
        id = fetchNode(id).nextNode
    Loop
errrtn:
    fsd_getChildNodes = nodesFound

End Function

Public Function fsd_getRootNode() As Integer
    Dim id As Integer
    Do While id <> -1

        If fetchNode(id).typeNode = NODE_ELEMENT Then
            fsd_getRootNode = id
            Exit Function
        End If
        id = fetchNode(id).nextNode
    Loop
errrtn:
    fsd_getRootNode = 0
End Function


Function fsd_getNodeName(nodeId As Integer) As String
    fsd_getNodeName = fsd_getText(fetchNode(nodeId).locName)
End Function


Function fsd_getAttributeCount(nodeId As Integer) As Integer
    Dim id As Integer
    Dim cnt As Integer
    On Error GoTo errrtn
    id = fetchNode(nodeId).firstAttribute

    Do While id <> -1
```

```
        cnt = cnt + 1
        id = fetchAttribute(id).nextAttribute
    Loop
errrtn:
    fsd_getAttributeCount = cnt
End Function


Function fsd_getNthAttribute(nodeId As Integer, attributeNum As Integer) As Integer
    Dim id As Integer
    Dim cnt As Integer
    On Error GoTo errrtn
    id = fetchNode(nodeId).firstAttribute

    Do While id <> -1
        If cnt = attributeNum Then
            fsd_getNthAttribute = id
            Exit Function
        End If
        cnt = cnt + 1
        id = fetchAttribute(id).nextAttribute
    Loop
errrtn:
    fsd_getNthAttribute = id
End Function

Function fsd_getAttribute(nodeId As Integer, sName As String) As String
    Dim attributeId As Integer
    Dim sNull As String

    On Error Resume Next
    attributeId = fsd_getAttributeByName(nodeId, sName)
    If attributeId <> -1 Then
        fsd_getAttribute = fsd_getAttributeValue(attributeId)
    Else
        fsd_getAttribute = sNull
    End If
End Function

Function fsd_getAttributeByName(nodeId As Integer, sName As String) As Integer
    Dim id As Integer
    Dim cnt As Integer
    On Error GoTo errrtn
    id = fetchNode(nodeId).firstAttribute

    Do While id <> -1
        If sName = fsd_getAttributeName(id) Then
            fsd_getAttributeByName = id
            Exit Function
        End If
        cnt = cnt + 1
        id = fetchAttribute(id).nextAttribute
    Loop
errrtn:
    fsd_getAttributeByName = -1
End Function


Function fsd_getAttributeName(nodeId As Integer) As String
    On Error GoTo errrtn
    fsd_getAttributeName = fsd_getText(fetchAttribute(nodeId).locName)
    Exit Function
errrtn:
    fsd_getAttributeName = Null
End Function
```

```
Function fsd_getAttributeValue(nodeId As Integer) As String
    On Error GoTo errrtn
    fsd_getAttributeValue = fsd_getText(fetchAttribute(nodeId).locValue)
    Exit Function
errrtn:
    fsd_getAttributeValue = Null
End Function

Function fsd_hasAttributes(nodeId As Integer) As Boolean
    On Error Resume Next
    fsd_hasAttributes = fetchNode(nodeId).firstAttribute <> -1
End Function


Sub fsd_readFile(filename As String)
    Dim i As Integer

    Open filename For Binary As #1
    Get #1, 1, header

    ReDim nodes(header.numNodes - 1)
    numNodes = header.numNodes
    Get #1, , nodes

    ReDim attributes(header.numAttributes - 1)
    numAttributes = header.numAttributes
    Get #1, , attributes

    ReDim textBuffer(header.lenTextArea - 1)
    Get #1, , textBuffer
    nextTextLoc = header.lenTextArea

    Close #1

End Sub



Function fsd_setAttribute(parentNode As Integer, name As String, value As String) As Integer
    Dim sName As String, sValue As String
    Dim attrId As Integer
    Dim parentAttrId As Integer, attrCount As Integer
    Dim node As node_def
    Dim localAttr As attribute_def

    On Error GoTo errrtn
    If parentNode >= 0 Then

        fsd_setAttribute = -1
        Exit Function
    End If

    attrId = fsd_getAttributeByName(parentNode, name)
    If attrId <> -1 Then


        localAttr = fetchAttribute(attrId)

        fsd_scratchTextBlock localAttr.locValue
        localAttr.locValue = -1

        localAttr.locValue = fsd_addText(value)
        saveAttribute attrId, localAttr
        fsd_setAttribute = attrId
        Exit Function
```

```
    End If


    attrId = fsd_slotAttribute()
    localAttr = fetchAttribute(attrId)

    localAttr.parentNode = parentNode
    localAttr.nextAttribute = -1


    localAttr.locName = fsd_addText(name)
    localAttr.locValue = fsd_addText(value)

    saveAttribute attrId, localAttr
    fsd_setAttribute = attrId


    attrCount = fsd_getAttributeCount(parentNode)
    If attrCount = 0 Then

        node = fetchNode(parentNode)
        node.firstAttribute = attrId
        saveNode parentNode, node
    Else

        parentAttrId = fsd_getNthAttribute(parentNode, attrCount - 1)
        localAttr = fetchAttribute(parentAttrId)
        localAttr.nextAttribute = attrId
        saveAttribute parentAttrId, localAttr
    End If
    Exit Function
errrtn:

End Function
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#ifndef __mainlinepic_h_
#define __mainlinepic_h_

void pwm_osc_init(unsigned long pwm_osc_frequency, unsigned short pwm_osc_dutycycle);
void pwm_stop(void);
void pwm_start(void);
void DisplayErrorMessageV(char *str);
void DisplayErrorMessageC(const char *str);

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "fsdinterpretertable.h"
#ifdef IR_RULES
#include "sendircommon.h"
#include "sendirrules.h"
#endif
#ifdef IR_UNIV_CHIP
#include "sendirunivchip.h"
#endif
#include "eprom.h"
#include "beep.h"
extern char               TEMPBUFFER[];
extern short      currentScriptBuffer;


static const char *commands[42] = {"TopMenu", "TitleMenu", "Resume", "Back",
          "Play", "Stop", "Pause", "Next", "Previous",    "TitleSeek",    "ChapterSeek",
          "language",    "StepForward",    "StepReverse",    "FastForward",    "FastReverse",
          "Set",    "Get",    "ButtonsOnInternal",    "ButtonsOffInternal",    "Restart",
          "GetIrScript",    "SetIrScript",    "Append",    "",    "",    "If",
          "Button",    "TrickPlay",    "Sleep",    "TimeSeek",    "Increment",
          "SleepHard",    "PushButton",    "PushNumbers",    "ButtonsOn",    "ButtonsOff",
          "IrRaw", "IrRawPart", "IrSend",
          NULL };

static short procCommand(short iCommand, NodeId commandNode, NodeId buttons[], short len);
static void info(const char *msg);


void PushPlayInitialize(void)
{
        fsdint_initCommands(commands, procCommand, info);
}


static short procCommand(short iCommand, NodeId node, NodeId buttons[], short len)
{
        TextLoc loc1, loc2, loc3;
        PtrTextLoc sValue, sValue2, sCommand;
        int ticks;

        short iMinValue, iMaxValue, count, iSecond, iMillisecond;

        loc1 = fsd_slotTextBlock();
        loc2 = fsd_slotTextBlock();
        loc3 = fsd_slotTextBlock();

        sValue = fsd_fetchTextLocPtr(loc1);
        sValue2 = fsd_fetchTextLocPtr(loc2);
        sCommand = fsd_fetchTextLocPtr(loc3);

        count = 0;

        if (sValue != NULL && sValue2 != NULL && sCommand != NULL) {

                ticks = GetTicks()& 0x7FFF;
                fsd_getNodeName(node, sCommand, CHAR_BUFFERSIZE);
                if (iCommand != 39) {
```

```
        debug(("%d Command %s %d:", ticks, sCommand, currentScriptBuffer ))
}
switch (iCommand) {
                case 0:
                        ir_sendWords(MENU) ;
                        break;
                case 1:
                        ir_sendWords(TITLE) ;
                        break;
                case 2:
                        ir_sendWords(PLAY) ;
                        break;
                case 3:
                        ir_sendWords(MENU) ;
                        break;
                case 4:

                        ir_sendWords(PLAY) ;
                        break;
                case 5:
                        ir_sendWords(STOPDVD) ;
                        break;
                case 6:
                        ir_sendWords(PAUSE) ;
                        break;
                case 7:
                        ir_sendWords(NEXTCHAPTER) ;
                        break;
                case 8:
                        ir_sendWords(PREVCHAPTER) ;
                        break;
                case 9:
        fsdint_fetch("title", sValue, CHAR_BUFFERSIZE);
                        debugHi(("%s", sValue));

                        buttons[0] = ir_findMacro(TITLESEEK, "TITLESEEK")
                        if (buttons[0] != NODE_ERROR) count = 1;
                        break;
                case 10:
        fsdint_fetch("chapter", sValue, CHAR_BUFFERSIZE);
                        debugHi(("%s", sValue));

                        buttons[0] = ir_findMacro(CHAPTERSEEK, "CHAPTERSE
                        if (buttons[0] != NODE_ERROR) count = 1;
                        break;
                case  16:
                        fsdint_fetch("id", sValue, CHAR_BUFFERSIZE);
                        fsdint_fetch("value", sValue2, CHAR_BUFFERSIZE);
                        debugHi(("%s=%s", sValue ,sValue2));
                        fsdint_store (sValue, sValue2);
                        break;
                case 18:
                        fsdint_ButtonsOnInternal();
                        break;
                case 19:
                        fsdint_ButtonsOffInternal();
                        break;
                case 20:
                        fsdint_Restart();
                        break;
                case 21:
                        fsdint_Reset();
                        break;
                case 22:
                        fsdint_GetIrScript();
                        break;
                case 23:
```

```
                    fsdint_fetch("value", sValue, CHAR_BUFFERSIZE);
iSecond = fsd_getInteger(sValue);
                    fsdint_SetIrScript(iSecond);
                    break;
          case 24:
                    fsdint_fetch("id", sValue, CHAR_BUFFERSIZE);
                    fsdint_fetch("value", sValue2, CHAR_BUFFERSIZE);
                    fsdint_append (sValue, sValue2);
                    fsdint_fetch(sValue, sValue2, CHAR_BUFFERSIZE);
                    keypressBeep();
                    debugHi(("%s=%s", sValue ,sValue2));
                    break;
          case 30:
                    fsdint_fetch("seconds", sValue, CHAR_BUFFERSIZE);
iSecond = fsd_getInteger(sValue);
                    fsdint_fetch("milliseconds", sValue, CHAR_BUFFERS
                    iMillisecond = fsd_getInteger(sValue);
                    debugHi(( "%d %d",  iSecond, iMillisecond ));
                    fsdint_delay( iSecond, iMillisecond);
                    break;

          case 31:
fsdint_fetch("time", sValue, CHAR_BUFFERSIZE);
                    debugHi(("%s",sValue));

                    buttons[0] = ir_findMacro(TIMESEEK, "TIMESEEK");
                    if (buttons[0] != NODE_ERROR) count = 1;
                    break;

          case 32:
                    fsdint_fetch("id", sValue, CHAR_BUFFERSIZE);
                    fsdint_fetch("min", sValue2, CHAR_BUFFERSIZE);
                    iMinValue = fsd_getInteger(sValue2);
                    fsdint_fetch("max", sValue2, CHAR_BUFFERSIZE);
                    iMaxValue = fsd_getInteger(sValue2);
                    debugHi(("%s %d %d", sValue, iMinValue, iMaxValue
                    fsdint_increment (sValue, iMinValue, iMaxValue);
                    break;
          case 33:
                    fsdint_fetch("seconds", sValue, CHAR_BUFFERSIZE);
iSecond = fsd_getInteger(sValue);
                    fsdint_fetch("milliseconds", sValue, CHAR_BUFFERS
                    iMillisecond = fsd_getInteger(sValue);
                    debugHi(("%d %d",  iSecond, iMillisecond ));
                    fsdint_hardDelay( iSecond, iMillisecond);
                    break;
          case 34:
                    fsdint_fetch("id", sValue, CHAR_BUFFERSIZE);
                    debugHi(("%s", sValue));

                    iSecond = (short)ir_lookupButton(sValue);
                    ir_sendWords((char)iSecond) ;
                    break;
          case 35:
                    fsdint_fetch("value", sValue, CHAR_BUFFERSIZE);
                    debugHi(("%s", sValue));

                    ir_sendNumbersString(sValue);
                    break;
          case 36:
                    fsdint_ButtonsOn();
                    break;
          case 37:
                    fsdint_ButtonsOff();
                    break;
          case 38:
                    break;
```

```
                                case 39:
                                        break;
                                case 40:
                                        break;
                        default:
                                        debugPutstrHi("Command not implemented");
                }
        }
        else {
                info("No textLoc avail");
        }

        fsd_scratchTextBlock(loc1);
        fsd_scratchTextBlock(loc2);
        fsd_scratchTextBlock(loc3);
        return count;
}

static void info(const char *msg)
{
        debugPutstrHi(msg);
}
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */
#include "support.h"

#include "rstack.h"
#ifdef DEBUG
#include <stdio.h>
#endif

#include <string.h>
#include "fsdtablelarge.h"

#ifdef PIC
#include "delay.h"
#include "i2c_ccs.h"
#include "tablereadwrite.h"
#else
#include "pcromchip.h"
#endif
#include "eprom.h"
#include "beep.h"


const char EMPTY_STRING[] = "";

extern char TEMPBUFFER[];
extern short devTicks;



extern const unsigned char *flashMemory;



static struct header_def header[NUMSCRIPTS];

#ifdef PIC
near
#endif

unsigned short    scriptBuffer[NUMSCRIPTS];
#ifdef PIC
near
#endif
short                                       numScriptBuffers;

#ifdef PIC
near
#endif
short                                       currentScriptBuffer;

#ifdef PIC
near
#endif
short                                       irScriptBuffer;

#ifdef PIC
near
#endif
short                                       mainScriptBuffer;
```

```
#ifdef PIC
near
#endif
short                   offsetFlashMemory = 0;


struct node_def                 dynamicNodes[NUMDYNAMICNODES];
short                                   maxNode;
struct attribute_def    dynamicAttributes[NUMDYNAMICATTRIBUTES];
short                                   maxAttribute;
char                                    dynamicTextBuffer[SIZETEXTBUFFER];
TextLoc                                 maxTextLoc;


static void *fetchNodePtr(const NodeId nodeId, const short offset);

static void *fetchAttributePtr(const NodeId nodeId, const short offset);


void fsd_Initialize(void)
{
        short i;
        for (i=0; i < NUMSCRIPTS; i++) {
                scriptBuffer[i] = 0;
        }
#ifndef PIC
        pc_Init();
#endif

        epromInitialize(FALSE);

        numScriptBuffers = 0;
        currentScriptBuffer = 0;
        offsetFlashMemory = 0;
}


void fsd_LoadMainScript(void)
{
        struct eprom_script_def script;
        short scriptType, scriptId;

        scriptType = MAINSCRIPT;

        if (devTicks == -1) {
                scriptId = IRGETSCRIPTID;
        }
        else {
                scriptId = -1;
        }
        script.location = -1;

        if (epromGetScript(scriptType, scriptId, &script) == -1) {
                fsd_setScriptBuffer(scriptType, scriptId);
        } else {
                fsd_setScriptBufferNoLoad(&script);
        }
}



void fsd_setMainScriptBuffer(void)
{
        currentScriptBuffer = mainScriptBuffer;
}
```

```
void fsd_switchRomBuffer(short newRomBuffer)
{
        RPush(currentScriptBuffer);
        currentScriptBuffer = newRomBuffer;
}


void fsd_unswitchRomBuffer()
{
        currentScriptBuffer = RPop();
}




static void *fetchNodePtr(const NodeId nodeId, const short offset)
{
        NodeId node;
        long address;

        if (nodeId < 0 ) {
                node = abs(nodeId) - 2;
                if (node >= NUMDYNAMICNODES || node < 0) return (void *)NODE_ERROR;
                return (unsigned char *)&dynamicNodes[node] + offset;
        } else {
                node = nodeId;
                if (node >= header[currentScriptBuffer].numNodes || node < 0) {
                        return (void *)NODE_ERROR;
                }

                address =  scriptBuffer[currentScriptBuffer];
                address += header[currentScriptBuffer].nodeOffset;
                address += node * sizeof(struct node_def);
                address += offset;
                return (void *)address;
        }
}


static void *fetchAttributePtr(const NodeId nodeId, const short offset)
{
        NodeId node;
        long address;

        if (nodeId < 0 ) {
                node = abs(nodeId) - 2;
                if (node >= NUMDYNAMICATTRIBUTES || node < 0) return (void *)NODE_ERROR;
                return (unsigned char *)&dynamicAttributes[node] + offset;
        } else {
                node = nodeId;
                if (node >= header[currentScriptBuffer].numAttributes || node < 0) return (v

                address =  scriptBuffer[currentScriptBuffer];
                address += header[currentScriptBuffer].attributeOffset;
                address += node * sizeof(struct attribute_def);
                address += offset;
                return (void *)address;
        }
}
```

```
void *fsd_fetchTextLocPtr(const TextLoc locText)
{
    TextLoc thisLoc;
         long address;

    if (locText < 0) {
        thisLoc = abs(locText) - 2;
                  if (thisLoc < 0 || thisLoc >= SIZETEXTBUFFER - 1) return (void *)NODE_ERROR;
                  return (unsigned char *)&dynamicTextBuffer[thisLoc];
         }
         else {
                  thisLoc = locText;
                  if (thisLoc < 0 || thisLoc >= header[currentScriptBuffer].lenTextArea - 1) r

                  address =  scriptBuffer[currentScriptBuffer];
                  address += header[currentScriptBuffer].textAreaOffset;
                  address += thisLoc;
                  return (void *)address;
         }
}


NodeId fsd_fetchNode(PtrNode pNode, NodeId node)
{
         void *address;
         address = fetchNodePtr(node, 0);
         if (address == (void *)NODE_ERROR) return NODE_ERROR;

         if (node < 0) {

                  memcpy(pNode, address, sizeof(Node));
         }
         else {

#ifdef SCRIPT_IN_FLASH
                  memcpy(pNode, flashMemory+(long)address, sizeof(Node));
#else
                  ROM_Read((int)address, (char *)pNode, sizeof(Node));
#endif
         }
         return node;
}


NodeId fsd_fetchNodeId(const NodeId node, const short offset)
{
         unsigned char*address;
         NodeId   word;

         address = fetchNodePtr(node, offset);
         if (address == (void *)NODE_ERROR) return NODE_ERROR;
         if (node < 0) {

                  memcpy(&word, address, sizeof(WORD));
         }
         else {

#ifdef SCRIPT_IN_FLASH
                  memcpy(&word, flashMemory+(long)address, sizeof(WORD));
#else
                  word = ROM_ReadWord((int)address);
#endif

         }
         return word;
```

```
}


NodeId fsd_fetchAttribute(PtrAttribute pAttribute, NodeId attribute)
{
        void *address;
        address = fetchAttributePtr(attribute, 0);
        if (address == (void *)NODE_ERROR) return NODE_ERROR;
        if (attribute < 0) {

                memcpy(pAttribute, address, sizeof(Attribute));
        }
        else {

#ifdef SCRIPT_IN_FLASH
                memcpy(pAttribute, flashMemory+(long)address, sizeof(Attribute));
#else
                ROM_Read((int)address, pAttribute, sizeof(Attribute));
#endif
        }
        return attribute;
}


NodeId fsd_fetchAttributeId(const NodeId attribute, const short offset)
{
        unsigned char*address;
        WORD     word;

        address = fetchAttributePtr(attribute, offset);
        if (address == (void *)NODE_ERROR) return NODE_ERROR;

        if (attribute < 0) {

                memcpy(&word, address, sizeof(WORD));
        }
        else {

#ifdef SCRIPT_IN_FLASH
                memcpy(&word, flashMemory+(long)address, sizeof(WORD));
#else
                word = ROM_ReadWord((int)address);
#endif
        }

        return word;
}


TextLoc fsd_fetchNodeTextLoc(const NodeId node, const short offset)
{
        unsigned char *address;
        WORD     word;

        address = fetchNodePtr(node, offset);
        if (address == (void *)NODE_ERROR) return (TextLoc)NODE_ERROR;

        if (node < 0) {

                memcpy(&word, address, 2);
        }
        else {

#ifdef SCRIPT_IN_FLASH
                memcpy(&word, flashMemory+(long)address, sizeof(WORD));
#else
```

```
                word = ROM_ReadWord((int)address);
#endif
        }

        return word;
}


TextLoc fsd_fetchAttributeTextLoc(const NodeId attribute, const short offset)
{
        unsigned char *address;
        WORD     word;

        address = fetchAttributePtr(attribute, offset);
        if (address == (void *)NODE_ERROR) return (TextLoc)NODE_ERROR;

        if (attribute < 0) {

                memcpy(&word, address, 2);
        }
        else {

#ifdef SCRIPT_IN_FLASH
                memcpy(&word, flashMemory+(long)address, sizeof(WORD));
#else
                word = ROM_ReadWord((int)address);
#endif
        }

        return word;
}


void fsd_fetchText(TextLoc textLoc, short textLen, char *buffer, const short len)
{
        PtrTextLoc loc;
        short  size;

        loc = fsd_fetchTextLocPtr(textLoc);
        if (loc == (PtrTextLoc)NODE_ERROR) {
                strcpy(buffer, EMPTY_STRING);
                return;
        }

        if (textLen >= len)
                size = len - 1;
        else
                size = textLen;

        if (textLoc < 0) {

                strncpy(buffer, loc, size);
        } else {

#ifdef SCRIPT_IN_FLASH
                strncpy(buffer, flashMemory+(long)loc, size);
#else
                ROM_Read((int)loc, buffer, (char)size);
#endif
        }
        buffer[size] = 0;
}


NodeId fsd_slotAttribute(void)
{
        short i;
```

```
        for (i=0; i < NUMDYNAMICATTRIBUTES; i++) {
                if (dynamicAttributes[i].parentnode == NODE_AVAILABLE ) {

           if (i > maxAttribute) {
                                maxAttribute = i;
                        }
           dynamicAttributes[i].parentnode = NODE_ALLOCATED;
           dynamicAttributes[i].locname = 0;
           dynamicAttributes[i].locvalue = 0;
           dynamicAttributes[i].nextattribute = NODE_EMPTY;
           return (NodeId) -(i + 2);
                }
        }
        return NODE_ERROR;
}


void fsd_scratchAttribute(const NodeId nodeId)
{
        PtrAttribute pAttrib;


    if (nodeId < 0 )  {
                pAttrib = fetchAttributePtr(nodeId,0);
                if (pAttrib == (PtrAttribute)NODE_ERROR) return;
        fsd_scratchTextBlock (pAttrib->locname);
        fsd_scratchTextBlock (pAttrib->locvalue);
        pAttrib->parentnode = NODE_AVAILABLE;
    }
}


NodeId fsd_slotNode(void)
{
        short i;
        for (i=0; i < NUMDYNAMICNODES; i++) {
                if (dynamicNodes[i].parentnode == NODE_AVAILABLE ) {

           if (i > maxNode) maxNode = i;

                        dynamicNodes[i].typenode = NODE_ALLOCATED;
                        dynamicNodes[i].firstattribute = NODE_EMPTY;
                        dynamicNodes[i].firstchild = NODE_EMPTY;
                        dynamicNodes[i].locname = -1;
                        dynamicNodes[i].lenname = 0;
                        dynamicNodes[i].nextnode = NODE_EMPTY;
                        dynamicNodes[i].parentnode = NODE_EMPTY;

                        return (NodeId) -(i + 2);
                }
        }
        return NODE_ERROR;
}


void fsd_scratchNode(const NodeId nodeId)
{
        PtrNode pNode;
        short pos;
        NodeId attrib;

        pos = 0;

    if (nodeId < 0 )  {
                pNode = fetchNodePtr(nodeId,0);
                if (pNode == (PtrNode)NODE_ERROR) return;
```

```
                while ((attrib = fsd_getAttributeByPos(nodeId, pos)) != NODE_ERROR) {
                        fsd_scratchAttribute (attrib);
                        pos++;
                }

                fsd_scratchTextBlock (pNode->locname);
        pNode->parentnode = NODE_AVAILABLE;
    }
}


TextLoc fsd_slotTextBlock(void)
{
    TextLoc loc=0;

    while (loc < SIZETEXTBUFFER && (loc + 1) < SIZETEXTBUFFER) {
        if (dynamicTextBuffer[loc] == 0 && dynamicTextBuffer[loc + 1] == 0) {

            if (loc > maxTextLoc) {
                                maxTextLoc = loc;
                        }

                        dynamicTextBuffer[loc+1] = 1;
            return  -(loc + 2);
        }
        loc += TEXT_CHUNK;
    }
    return TEXTLOC_EMPTY;
}


void fsd_scratchTextBlock(const TextLoc loc)
{
         PtrTextLoc pText;

    if (loc < 0) {
                pText = fsd_fetchTextLocPtr(loc);
                if (pText == (PtrTextLoc)NODE_ERROR) return;
        *pText++ = 0;
        *pText = 0;
    }
}




short fsd_getAttributes(const NodeId parentNode, NodeId nodesFound[], const short len) {
    NodeId id;
    short cntNodesFound=0;

         id = fsd_fetchNodeId(parentNode, FIRSTATTRIBUTE);

    while (!(id == NODE_EMPTY || id == NODE_ERROR) ) {
                if (cntNodesFound >= len) return len;
        nodesFound[cntNodesFound] = id;
        cntNodesFound++;
        id = fsd_fetchAttributeId(id, NEXTATTRIBUTE);
    }
         return cntNodesFound;

}


void fsd_setNodeName(const NodeId node, const NodeId parent, const char *name)
```

```
{
        PtrNode pNode;
        PtrTextLoc pText;

        pNode = fetchNodePtr(node,0);

        pText = fsd_fetchTextLocPtr(pNode->locname);
        if (pText != (PtrTextLoc)NODE_ERROR) {
                fsd_scratchTextBlock(pNode->locname);
        }
        pNode->locname = fsd_addText(name);
        pNode->parentnode = parent;
        pNode->typenode = NODE_ELEMENT;
}



TextLoc fsd_addText(const char *sText)
{
    short slen;
    TextLoc loc;
    PtrTextLoc pText;

    slen = strlen(sText);
    if (slen == 0) return 0;

         if (slen > TEXT_CHUNK - 1 ) slen = TEXT_CHUNK - 1;

    loc = fsd_slotTextBlock();
         pText = fsd_fetchTextLocPtr(loc);
    if (pText == (PtrTextLoc)NODE_ERROR) {
        debugPutstrHi("addText err");
        return 0;
         }
    strncpy(pText,sText,slen);
         pText += slen;
         *pText = '\0';

    return loc;
}









NodeId fsd_getRootNode(void)
{
```

```
    NodeId id=0;
    while (!(id == NODE_EMPTY || id == NODE_ERROR)) {

                if (fsd_fetchNodeId(id,TYPENODE) == NODE_ELEMENT) {
            return id;
        }
        id = fsd_fetchNodeId(id, NEXTNODE);
    }
         return NODE_ERROR;
}




short fsd_getChildNodes(const NodeId parentNode, NodeId nodesFound[], const short len)
{
    NodeId id;
    short cntNodesFound=0;

    id = fsd_fetchNodeId(parentNode,FIRSTCHILD);

    while (!(id == NODE_EMPTY || id == NODE_ERROR)) {
                if (cntNodesFound >= len) return len;
        nodesFound[cntNodesFound] = id;
        cntNodesFound++;
        id = fsd_fetchNodeId(id, NEXTNODE);
    }
         return cntNodesFound;

}


NodeId fsd_getChildByPos(const NodeId parentNode, const short pos)
{
    NodeId id;
    short cnt=0;
    id = fsd_fetchNodeId(parentNode,FIRSTCHILD);

    while (!(id == NODE_EMPTY || id == NODE_ERROR) ) {
                if (cnt == pos) return id;
        cnt++;
        id = fsd_fetchNodeId(id, NEXTNODE);
    }
         return NODE_ERROR;
}


short fsd_getChildCount(const NodeId parentNode)
{
    NodeId id;
    short cntNodesFound=0;

    id = fsd_fetchNodeId(parentNode,FIRSTCHILD);

    while (!(id == NODE_EMPTY || id == NODE_ERROR) ) {
        cntNodesFound++;
        id = fsd_fetchNodeId(id, NEXTNODE);
    }
         return cntNodesFound;

}


void fsd_getNodeName(const NodeId nodeId, char *buffer, const short len)
```

```
{
        Node node;
        NodeId id;

        id = fsd_fetchNode(&node, nodeId);
        if (id == NODE_ERROR) {
                strcpy(buffer, EMPTY_STRING);
        }
        else {
                fsd_fetchText(node.locname, node.lenname, buffer, len);
        }
}


short fsd_getNodesByName(const NodeId parentNode, const char *sName, NodeId nodesFound[], con
{
    NodeId id;
    short cntNodesFound=0;

    id = fsd_fetchNodeId(parentNode,FIRSTCHILD);

    while (!(id == NODE_EMPTY || id == NODE_ERROR) ) {
                fsd_getNodeName(id, TEMPBUFFER, CHAR_BUFFERSIZE);
                if (strcmp(TEMPBUFFER,sName) == 0) {
                        nodesFound[cntNodesFound] = id;
                        cntNodesFound++;
                }
        id = fsd_fetchNodeId(id, NEXTNODE);
    }
        return cntNodesFound;
}


NodeId fsd_getAttributeByName(const NodeId parentNode, const char *sName)
{
    NodeId id;

        Attribute attrib;
        char count;

        count = 0;
    id = fsd_fetchNodeId(parentNode,FIRSTATTRIBUTE);

    while (!(id == NODE_EMPTY || id == NODE_ERROR)  ) {
                id = fsd_fetchAttribute(&attrib, id);
                if (id == NODE_ERROR) break;


                fsd_fetchText(attrib.locname, attrib.lenname, TEMPBUFFER, CHAR_BUFFERSIZE);
                if (strcmp(TEMPBUFFER,sName) == 0) {
                        return id;
                }

                id = attrib.nextattribute;
            if (count++ > 100) break;
    }
        return NODE_ERROR;
}


short fsd_getAttributeCount(const NodeId parentNode)
{
    NodeId id;
        short cnt=0;

    id = fsd_fetchNodeId(parentNode,FIRSTATTRIBUTE);
```

```
    while (!(id == NODE_EMPTY || id == NODE_ERROR)) {
                cnt++;
            id = fsd_fetchAttributeId(id,NEXTATTRIBUTE);
    }
        return cnt;
}


NodeId fsd_getAttributeByPos(const NodeId parentNode, const short pos)
{
    NodeId id;
        short cnt=0;

    id = fsd_fetchNodeId(parentNode,FIRSTATTRIBUTE);

    while (!(id == NODE_EMPTY || id == NODE_ERROR)) {
                if (cnt == pos) {
                        return id;
                }
                cnt++;
            id = fsd_fetchAttributeId(id,NEXTATTRIBUTE);
    }
        return NODE_ERROR;
}


void fsd_getAttributeValue(const NodeId attributeId, char *buffer, const short len)
{
        Attribute attrib;
        NodeId id;


        id = fsd_fetchAttribute(&attrib, attributeId);
        if (id == NODE_ERROR) {
                strcpy(buffer, EMPTY_STRING);
                return;
        }



        fsd_fetchText(attrib.locvalue, attrib.lenvalue, buffer, len);

}


void fsd_getAttribute(const NodeId parentNode, const char *attribName, char *buffer, const sh
{
        NodeId attribNode;
    attribNode = fsd_getAttributeByName(parentNode, attribName);
        if (attribNode == NODE_ERROR) {
                strcpy(buffer, EMPTY_STRING);
                return;
        }
        fsd_getAttributeValue(attribNode, buffer, len);
}


BOOL fsd_hasAttributes(const NodeId nodeId)
{
        return fsd_fetchNodeId(nodeId, FIRSTATTRIBUTE) != NODE_EMPTY;
}


BOOL fsd_hasChildNodes(const NodeId nodeId)
{
        return fsd_fetchNodeId(nodeId, FIRSTCHILD) != NODE_EMPTY;
}
```

```
NodeId fsd_setAttribute(const NodeId parentNode, const char *name, const char *value)
{
    NodeId attrId;
         NodeId attrParent;
         short attrCount;
    PtrAttribute pAttribute;
         PtrNode pNode;


    if ( parentNode >= 0 ) {

        return NODE_ERROR;
    }
    attrId = fsd_getAttributeByName(parentNode, name);
    if ( attrId != NODE_ERROR ) {

        pAttribute = fetchAttributePtr(attrId,0);
                  if (pAttribute == (PtrAttribute)NODE_ERROR) return NODE_ERROR;

                  fsd_scratchTextBlock (pAttribute->locvalue);

                  pAttribute->locvalue = fsd_addText(value);
                  pAttribute->lenvalue = (unsigned char)strlen(value);
                  return attrId;
    }


    pNode = fetchNodePtr(parentNode,0);
         if (pNode == (PtrNode)NODE_ERROR) return NODE_ERROR;

    attrId = fsd_slotAttribute();
         if (attrId == NODE_ERROR) return NODE_ERROR;
    pAttribute = fetchAttributePtr(attrId,0);

    pAttribute->parentnode = parentNode;
    pAttribute->nextattribute = NODE_EMPTY;

    pAttribute->locname = fsd_addText(name);
         pAttribute->lenname = (unsigned char)strlen(name);
    pAttribute->locvalue = fsd_addText(value);
         pAttribute->lenvalue = (unsigned char)strlen(value);



         attrCount = fsd_getAttributeCount(parentNode);
    if ( attrCount == 0 ) {
        pNode->firstattribute = attrId;
    } else {

                  attrParent = fsd_getAttributeByPos(parentNode, (const short)(attrCount - 1))
        pAttribute = fetchAttributePtr(attrParent, 0);
                  if (pAttribute == (PtrAttribute)NODE_ERROR) return NODE_ERROR;
        pAttribute->nextattribute = attrId;
    }
    return attrId;
}


short fsd_getInteger(const char *value)
{

    return atoi(value);
```

```
}


static void readHeaderFlash(void)
{
        struct header_def *headerFrom;

        headerFrom = (struct header_def *)(flashMemory+scriptBuffer[currentScriptBuffer]);
        memcpy(&header[currentScriptBuffer], headerFrom, sizeof(struct header_def));
}



#ifdef SCRIPT_IN_FLASH
void fsd_moveScriptFlash(struct script_def *script, short numBytes)
{
#ifdef PIC
        short count, sourceOffset,destOffset,chunk;
        int romAddress;
        unsigned long destAddress;
        char buffer[32];




        numBytes = (numBytes + 7) & 0xfff8;



        count = 0;
        romAddress = script->location;
        destAddress = (unsigned long)flashMemory;

        sourceOffset = 0;
        destOffset = offsetFlashMemory;

        while(count < numBytes) {

                if ( count+32 < numBytes) {
                        chunk = 32;
                }
                else {
                        chunk = numBytes - count;
                }
                ROM_Read(romAddress+sourceOffset, buffer, chunk);
                TableWrite((unsigned char *)(destAddress+destOffset), buffer, chunk);









                count += chunk;
                sourceOffset += chunk;
                destOffset += chunk        ;
        }
#else

        pc_moveScriptFlash(script, numBytes);
```

```
#endif
}
#endif

#ifdef PIC

static void readHeaderROM(unsigned char *address)
{

        header[currentScriptBuffer].nodeOffset = ROM_ReadWord((int)address+0);
        header[currentScriptBuffer].numNodes = ROM_ReadWord((int)address+2);
        header[currentScriptBuffer].attributeOffset = ROM_ReadWord((int)address+4);
        header[currentScriptBuffer].numAttributes = ROM_ReadWord((int)address+6);
        header[currentScriptBuffer].textAreaOffset = ROM_ReadWord((int)address+8);
        header[currentScriptBuffer].lenTextArea = ROM_ReadWord((int)address+10);
        header[currentScriptBuffer].scriptType = ROM_ReadWord((int)address+12);
        header[currentScriptBuffer].scriptId = ROM_ReadWord((int)address+14);
}
#endif


short fsd_findScript(short scriptType, short scriptId, struct script_def *script)
{
#ifdef PIC
        unsigned char scriptFound;
        short i, numScripts, scriptOrigin;

        scriptOrigin = sizeof(struct control_def);
        numScripts = ROM_ReadWord(NUMBERSCRIPTS);
        scriptFound = FALSE;

        for (i=0; i < numScripts; i++) {
                script->type = ROM_ReadWord(0 + (i * sizeof(struct script_def) + scriptOrigi
                script->id = ROM_ReadWord(2 + (i * sizeof(struct script_def) + scriptOrigin)
                script->location = ROM_ReadWord(4 + (i * sizeof(struct script_def) + scriptC

                if (script->type != scriptType) continue;
                if (scriptId != -1 && script->id != scriptId) continue;
                readHeaderROM((unsigned char *)script->location);
                scriptFound = TRUE;
                break;
        }
        return scriptFound;

#else
        return pc_findScript(scriptType, scriptId, script, &header[currentScriptBuffer]);
#endif
}



void fsd_setScriptBuffer(short scriptType, short scriptId)
{
#ifdef SCRIPT_IN_FLASH
        struct eprom_script_def epromScript;
        short numBytes;
#endif
        struct script_def script;
        short scriptFound;
        short saveScriptBuffer;

        if (!(numScriptBuffers < NUMSCRIPTS)) {
                errorBeep();
                debugPutstrHi("too many scripts");
                return;
```

```
        }

        saveScriptBuffer = currentScriptBuffer;
        currentScriptBuffer = numScriptBuffers;

        scriptFound = fsd_findScript(scriptType, scriptId, &script);
        if (scriptFound) {
                debug(("Script Found %d %d", script.type, script.id));
                scriptBuffer[currentScriptBuffer] = script.location;
                numScriptBuffers++;
                if (scriptType == IRSCRIPT) {
                        irScriptBuffer = currentScriptBuffer;
                }

                if (scriptType == MAINSCRIPT) {
                        mainScriptBuffer = currentScriptBuffer;
                }

#ifdef SCRIPT_IN_FLASH


                numBytes = sizeof(struct header_def);
                numBytes += header[currentScriptBuffer].numNodes * sizeof(struct node_def);
                numBytes += header[currentScriptBuffer].numAttributes * sizeof(struct attrit
                numBytes += header[currentScriptBuffer].lenTextArea;

                if ( (numBytes + offsetFlashMemory) < FLASHAREASIZE) {
                        fsd_moveScriptFlash(&script, numBytes);
                        scriptBuffer[currentScriptBuffer] = offsetFlashMemory;
                        offsetFlashMemory += numBytes;


                        epromScript.id = header[currentScriptBuffer].scriptId;
                        epromScript.location = scriptBuffer[currentScriptBuffer];
                        epromScript.type = header[currentScriptBuffer].scriptType;
                        epromScript.len = numBytes;
                        epromWriteScriptNumber(currentScriptBuffer, &epromScript);
                }

#endif
        }
        else {
                currentScriptBuffer = saveScriptBuffer;

                if (scriptType == IRSCRIPT) {
                        epromWriteWord(EPROM_IR_SCRIPTID, -1);
                }
                debug(("Script Type: %d Id: %d Not Found", scriptType, scriptId));
                errorBeep();
        }

}



void fsd_setScriptBufferNoLoad(struct eprom_script_def *script)
{

        currentScriptBuffer = numScriptBuffers;
        numScriptBuffers++;
        offsetFlashMemory = script->location + script->len;
        if (script->type == IRSCRIPT) {
                irScriptBuffer = currentScriptBuffer;
        } else if (script->type == MAINSCRIPT) {
                mainScriptBuffer = currentScriptBuffer;
        }
```

```
        scriptBuffer[currentScriptBuffer] = script->location;
#if !defined PIC && defined SCRIPT_IN_FLASH

        pc_readFlash(script->location, script->len);
#endif
        readHeaderFlash();
        debug(("RomScript %d %d %d %d", script->type, script->id, script->location, script->
}


void fsd_clearEpromScript(short scriptType, short scriptId)
{
        short i;
        struct eprom_script_def script;

        for (i = EPROM_NUM_SCRIPTS - 1; i > 0; i--) {
                epromGetScriptNumber(i, &script);
                if (script.type != -1) {
                        epromInitializeScript(i);

                        if (script.type != scriptType) continue;
                        if (scriptId != -1 && script.id != scriptId) continue;
                        numScriptBuffers--;
                        break;
                }
        }
}
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

 #include "support.h"

#ifdef PIC
#include <pic18.h>
#include "delay.h"
#else
#include <stdio.h>
#endif
#include "beep.h"


#ifdef PIC



void beep( int frequency, int duration )
{
        long totalTime, freq;


    if( duration < 75 )  duration = 75;
    totalTime = (long)(duration * 1000L);


        if (frequency == 0) {
                DelayBigUs(totalTime);
                return;
        }

    if( frequency < c0) frequency = c0;


    freq = (long)(1000000L / (frequency * 2));
    di();
        while (totalTime > 0 ) {
                BEEPER = 1;
                DelayBigUs(freq);
                totalTime -= freq;
                BEEPER = 0;
                DelayBigUs(freq);
                totalTime -= freq;
        }
        ei();
}
#endif

void goodBeep(void)
{
#ifdef PIC
        beep (c1, EIGHTH);
        beep (g1, EIGHTH);
#else
        printf("goodBeep\a");
#endif
}

void errorBeep(void)
```

```
{
        char i;

        for (i=0; i < 3; i++) {
#ifdef PIC
                beep (c2, EIGHTH);
                beep (e3, EIGHTH);
                beep (g2, EIGHTH);
                beep (c3, EIGHTH);
#else
                printf("errorBeep\a");
#endif
        }
}

void keypressBeep(void)
{
#ifdef PIC
        beep(c2, EIGHTH);
#else
        printf("\a");
#endif
}
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"

#ifdef DEBUG

#include <pic18.h>
#include "config.h"
#include <stdio.h>
#include "serial.h"


void
init_comms(void)
{









        TRISC6=OUTPUT;
        TRISC7=INPUT;
        SPBRG= SPBRG_DIVIDER;
        BRGH=1;
        SYNC=0;
        SPEN=1;
        SREN=0;
        TXIE=0;
        RCIE=0;
        TX9=0;
        RX9=0;
        TXEN=0;
        TXEN=1;
        CREN=0;
        CREN=1;
}

void
putch(unsigned char byte)
{

        while(!TXIF)
                continue;
        TXREG = byte;
}

unsigned char
getch() {

        while(!RCIF)
                continue;
        return RCREG;
}

unsigned char
getche(void)
```

```
{
        unsigned char c;
        putch(c = getch());
        return c;
}

char *getsNoEcho(char *s)
{
        register char *   s1 = s;
        int       c;

        for(;;) {
                switch(c = getch()) {

                        case '\n':
                        case '\r':
                                *s1 = 0;
                                return s;

                        default:
                                *s1++ = c;
                                break;
                }
        }
}

char *gets(char *s)
{
        register char *   s1 = s;
        int       c;

        for(;;) {
                switch(c = getche()) {

                        case '\n':
                        case '\r':
                                *s1 = 0;
                                return s;

                        default:
                                *s1++ = c;
                                break;
                }
        }
}


puts(const char *s)
{
        while(*s)
                putch(*s++);
        putch('\r');
        putch('\n');
}

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"
#include "squeue.h"

#define QUEUE_LENGTH QUEUE_DIM-1
#ifdef PIC
near
#endif
static int last=0;
#ifdef PIC
near
#endif
static int first=QUEUE_LENGTH;

static char val[QUEUE_DIM][MAXQUEUELENGTH];

char QueueIsFull(void)
{
        return (last>first ? last-first : QUEUE_DIM+last-first)>=QUEUE_DIM;
}

char QueueIsEmpty(void)
{
        return (last>first ? last-first : QUEUE_DIM+last-first) <= 1;
}


void EmptySQueue(void)
{
        last = 0;
        first = QUEUE_LENGTH;
}


void SEnqueue(const char *el)
{
        int slen;

        if (!QueueIsFull())        {
                slen = strlen(el);
                if (slen > (MAXQUEUELENGTH - 1))
                        slen = MAXQUEUELENGTH - 1;
                strncpy(val[last],el, slen);
                val[last][slen] = 0;
                last++;
                if(last>=QUEUE_DIM) last-=QUEUE_DIM;
        }
        else {
                debugPutstrHi("SQueue Full");
        }
}



char SDequeue(char *el, const int len)
{
        int slen;
        if (!QueueIsEmpty())       {
                if (++first>=QUEUE_DIM) first-=QUEUE_DIM;
```

```
                slen = strlen(val[first]);
                if (slen > (len - 1))
                        slen = len - 1;
                strncpy(el, val[first], slen);
                el[slen] = 0;
                return 1;
        }
        return 0;
}
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

#include "support.h"

#ifdef DEBUG

#include <pic18.h>
#include "config.h"
#include <stdio.h>
#include "serial.h"


void
init_comms(void)
{









        TRISC6=OUTPUT;
        TRISC7=INPUT;
        SPBRG= SPBRG_DIVIDER;
        BRGH=1;
        SYNC=0;
        SPEN=1;
        SREN=0;
        TXIE=0;
        RCIE=0;
        TX9=0;
        RX9=0;
        TXEN=0;
        TXEN=1;
        CREN=0;
        CREN=1;
}

void
putch(unsigned char byte)
{

        while(!TXIF)
                continue;
        TXREG = byte;
}

unsigned char
getch() {

        while(!RCIF)
                continue;
        return RCREG;
}

unsigned char
getche(void)
```

```
{
        unsigned char c;
        putch(c = getch());
        return c;
}

char *getsNoEcho(char *s)
{
        register char *   s1 = s;
        int       c;

        for(;;) {
                switch(c = getch()) {

                        case '\n':
                        case '\r':
                                *s1 = 0;
                                return s;

                        default:
                                *s1++ = c;
                                break;
                }
        }
}

char *gets(char *s)
{
        register char *   s1 = s;
        int       c;

        for(;;) {
                switch(c = getche()) {

                        case '\n':
                        case '\r':
                                *s1 = 0;
                                return s;

                        default:
                                *s1++ = c;
                                break;
                }
        }
}


puts(const char *s)
{
        while(*s)
                putch(*s++);
        putch('\r');
        putch('\n');
}

#endif
```

```
/*
 * PushPlay -- An Xml Document emulator\interpreter for microprocessors
 *
 * Copyright (C) 2002, Arthur Gravina. Confidential.
 *
 * Arthur Gravina <art@agravina.com>
 *
 */

 #include "support.h"

#ifdef PIC
#include <pic18.h>
#include "delay.h"
#else
#include <stdio.h>
#endif
#include "beep.h"


#ifdef PIC



void beep( int frequency, int duration )
{
        long totalTime, freq;


    if( duration < 75 )  duration = 75;
    totalTime = (long)(duration * 1000L);


        if (frequency == 0) {
                DelayBigUs(totalTime);
                return;
        }

    if( frequency < c0) frequency = c0;


    freq = (long)(1000000L / (frequency * 2));
    di();
        while (totalTime > 0 ) {
                BEEPER = 1;
                DelayBigUs(freq);
                totalTime -= freq;
                BEEPER = 0;
                DelayBigUs(freq);
                totalTime -= freq;
        }
        ei();
}
#endif

void goodBeep(void)
{
#ifdef PIC
        beep (c1, EIGHTH);
        beep (g1, EIGHTH);
#else
        printf("goodBeep\a");
#endif
}

void errorBeep(void)
```

```
{
        char i;

        for (i=0; i < 3; i++) {
#ifdef PIC
                beep (c2, EIGHTH);
                beep (e3, EIGHTH);
                beep (g2, EIGHTH);
                beep (c3, EIGHTH);
#else
                printf("errorBeep\a");
#endif
        }
}

void keypressBeep(void)
{
#ifdef PIC
        beep(c2, EIGHTH);
#else
        printf("\a");
#endif
}
```